教育部高等学校网络空间安全专业教学指导委员会
中国计算机学会教育专业委员会
共同指导



奇安信集团组织编写

网络空间安全重点规划丛书

顾问委员会主任：沈昌祥　编委会主任：封化民

代码安全实验指导

杨东晓　章磊　吴迪　司乾伟　编著

Cyberspace Security

根据教育部高等学校信息安全专业教学指导委员会编制的《高等学校信息安全专业指导性专业规范》组织编写

清华大学出版社

网络空间安全重点规划丛书

代码安全实验指导

杨东晓　章　磊　吴　迪　司乾伟　编著

清华大学出版社
北　京

内 容 简 介

本书为"代码安全"课程的实验教材。全书分为5章，主要内容包括代码安全保障系统基本配置、代码安全保障系统缺陷检测、代码安全保障系统合规检测、代码安全保障系统发起检测任务，以及代码安全保障系统检测结果审计。

本书由奇安信集团针对高校网络空间安全专业的教学规划组织编写，既可以作为高校信息安全、网络空间安全等相关专业的教材，及网络工程、计算机技术应用培训教材，也可以作为网络管理人员及对计算机网络、网络空间安全感兴趣的读者的基础读物。

图书在版编目(CIP)数据

代码安全实验指导/杨东晓等编著.—北京：清华大学出版社，2020.8
(网络空间安全重点规划丛书)
ISBN 978-7-302-55469-1

Ⅰ.①代…　Ⅱ.①杨…　Ⅲ.①软件开发－安全技术　Ⅳ.①TP311.522

中国版本图书馆CIP数据核字(2020)第086295号

责任编辑：张　民
封面设计：
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-83470235
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-83470236
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**18.5　　**字　　数：**436千字
版　　次：2020年8月第1版　　**印　　次：**2020年8月第1次印刷
定　　价：49.50元

产品编号：085313-01

网络空间安全重点规划丛书

编审委员会

出版说明

21世纪是信息时代，信息已成为社会发展的重要战略资源，社会的信息化已成为当今世界发展的潮流和核心，而信息安全在信息社会中将扮演极为重要的角色，它会直接关系到国家安全、企业经营和人们的日常生活。随着信息安全产业的快速发展，全球对信息安全人才的需求量不断增加，但我国目前信息安全人才极度匮乏，远远不能满足金融、商业、公安、军事和政府等部门的需求。要解决供需矛盾，必须加快信息安全人才的培养，以满足社会对信息安全人才的需求。为此，教育部继2001年批准在武汉大学开设信息安全本科专业之后，又批准了多所高等院校设立信息安全本科专业，而且许多高校和科研院所已设立了信息安全方向的具有硕士和博士学位授予权的学科点。

信息安全是计算机、通信、物理、数学等领域的交叉学科，对于这一新兴学科的培养模式和课程设置，各高校普遍缺乏经验，因此中国计算机学会教育专业委员会和清华大学出版社联合主办了"信息安全专业教育教学研讨会"等一系列研讨活动，并成立了"高等院校信息安全专业系列教材"编审委员会，由我国信息安全领域著名专家肖国镇教授担任编委会主任，指导"高等院校信息安全专业系列教材"的编写工作。编委会本着研究先行的指导原则，认真研讨国内外高等院校信息安全专业的教学体系和课程设置，进行了大量具有前瞻性的研究工作，而且这种研究工作将随着我国信息安全专业的发展不断深入。系列教材的作者都是既在本专业领域有深厚的学术造诣，又在教学第一线有丰富的教学经验的学者、专家。

该系列教材是我国第一套专门针对信息安全专业的教材，其特点是：

① 体系完整、结构合理、内容先进。

② 适应面广：能够满足信息安全、计算机、通信工程等相关专业对信息安全领域课程的教材要求。

③ 立体配套：除主教材外，还配有多媒体电子教案、习题与实验指导等。

④ 版本更新及时，紧跟科学技术的新发展。

在全力做好本版教材，满足学生用书的基础上，还经由专家的推荐和审定，遴选了一批国外信息安全领域优秀的教材加入系列教材中，以进一步满足大家对外版书的需求。"高等院校信息安全专业系列教材"已于2006年年初正式列入普通高等教育"十一五"国家级教材规划。

2007年6月，教育部高等学校信息安全类专业教学指导委员会成立大会

暨第一次会议在北京胜利召开。本次会议由教育部高等学校信息安全类专业教学指导委员会主任单位北京工业大学和北京电子科技学院主办，清华大学出版社协办。教育部高等学校信息安全类专业教学指导委员会的成立对我国信息安全专业的发展起到重要的指导和推动作用。2006年，教育部给武汉大学下达了“信息安全专业指导性专业规范研制”的教学科研项目。2007年起，该项目由教育部高等学校信息安全类专业教学指导委员会组织实施。在高教司和教指委的指导下，项目组团结一致，努力工作，克服困难，历时5年，制定出我国第一个信息安全专业指导性专业规范，于2012年年底通过经教育部高等教育司理工科教育处授权组织的专家组评审，并且已经得到武汉大学等许多高校的实际使用。2013年，新一届教育部高等学校信息安全专业教学指导委员会成立。经组织审查和研究决定，2014年，以教育部高等学校信息安全专业教学指导委员会的名义正式发布《高等学校信息安全专业指导性专业规范》(由清华大学出版社正式出版)。

2015年6月，国务院学位委员会、教育部出台增设“网络空间安全”为一级学科的决定，将高校培养网络空间安全人才提到新的高度。2016年6月，中央网络安全和信息化领导小组办公室(下文简称中央网信办)、国家发展和改革委员会、教育部、科学技术部、工业和信息化部及人力资源和社会保障部六大部门联合发布《关于加强网络安全学科建设和人才培养的意见》(中网办发文〔2016〕4号)。2019年6月，教育部高等学校网络空间安全专业教学指导委员会召开成立大会。为贯彻落实《关于加强网络安全学科建设和人才培养的意见》，进一步深化高等教育教学改革，促进网络安全学科专业建设和人才培养，促进网络空间安全相关核心课程和教材建设，在教育部高等学校网络空间安全专业教学指导委员会和中央网信办资助的网络空间安全教材建设课题组的指导下，启动了“网络空间安全重点规划丛书”的工作，由教育部高等学校网络空间安全专业教学指导委员会秘书长封化民教授担任编委会主任。本规划丛书基于“高等院校信息安全专业系列教材”坚实的工作基础和成果、阵容强大的编审委员会和优秀的作者队伍，目前已经有多本图书获得教育部和中央网信办等机构评选的“普通高等教育本科国家级规划教材”“普通高等教育精品教材”“中国大学出版社图书奖”和“国家网络安全优秀教材奖”等多个奖项。

“网络空间安全重点规划丛书”将根据《高等学校信息安全专业指导性专业规范》(及后续版本)和相关教材建设课题组的研究成果不断更新和扩展，进一步体现科学性、系统性和新颖性，及时反映教学改革和课程建设的新成果，并随着我国网络空间安全学科的发展不断完善，力争为我国网络空间安全相关学科专业的本科和研究生教材建设、学术出版与人才培养做出更大的贡献。

我们的E-mail地址是：zhangm@tup.tsinghua.edu.cn，联系人：张民。

“网络空间安全重点规划丛书”编审委员会

前言

没有网络安全，就没有国家安全；没有网络安全人才，就没有网络安全。

为了更多、更快、更好地培养网络安全人才，许多学校都在加大各方面投入，聘请优秀老师，招收优秀学生，建设一流的网络空间安全专业。

网络空间安全专业建设需要体系化的培养方案、系统化的专业教材和专业化的师资队伍。优秀教材是网络空间安全专业人才的关键。但是，这却是一项十分艰巨的任务。原因有二：其一，网络空间安全的涉及面非常广，至少包括密码学、数学、计算机、通信工程等多门学科，因此，其知识体系庞杂、难以梳理；其二，网络空间安全的实践性很强，技术发展更新非常快，对环境和师资要求也很高。

《代码安全实验指导》为《代码安全》一书的配套实验教材。通过实践教学，帮助学生理解和掌握C、C++、Java、PHP和Python语言中可能出现的安全漏洞，了解软件开发过程中如果不采取规范和安全的编码对系统造成的危害。培养学生养成规范和安全编码的习惯，并能够运用所学的技术和方法对企业软件开发过程中典型的软件安全问题进行安全分析、检测和安全加固。

本书分为5章。第1章介绍代码安全保障系统基本配置，第2章介绍代码安全保障系统缺陷检测，第3章介绍代码安全保障系统合规检测，第4章介绍代码安全保障系统发起检测任务，第5章介绍代码安全保障系统检测结果审计。

本书编写过程中得到奇安信集团的裴智勇、陈隆沛、童小刚和北京邮电大学雷敏等专家学者的鼎力支持，在此对他们的工作表示衷心的感谢！

本书适合作为高校网络空间安全、信息安全等相关专业的实验教材。随着新技术的不断发展，今后将不断更新图书内容。

由于作者水平有限，书中难免存在疏漏和不妥之处，欢迎读者批评指正。

作　者
2020年2月

目 录

第1章 代码安全保障系统基本配置

奇安信代码安全保障系统是奇安信企业安全集团基于多年源代码安全实践经验推出的新一代源代码安全检测解决方案，包括源代码安全缺陷检测、合规检测、溯源检测三大检测功能，同时奇安信代码安全保障系统还可实现软件安全开发生命周期管理，与企业已有代码版本管理系统(如SVN、GIT)、缺陷管理系统(如Bugzilla)等无缝对接，将源代码检测融入企业开发流程，实现软件源代码安全目标管理、自动化检测、差距分析、Bug修复追踪等功能，帮助企业以最小代价建立代码安全保障体系并落地实施，构筑信息系统的“内建安全”。

任何一个单位在购置代码安全保障系统设备时需要先完成基本的系统配置以及各个功能模块配置后才能使得系统稳定、高效地运行。本章主要完成代码安全保障系统的基本配置以及模块配置实验，通过这些实际操作，掌握系统配置、检测模板管理、日志管理及引擎管理等基础防护功能。

1.1 系统设置

本节主要完成代码安全保障系统配置实验。

【实验目的】

通过本实验使管理员熟练掌握对代码卫士进行邮箱配置及查看系统的授权状态。

【知识点】

邮箱配置、授权信息。

【场景描述】

A公司研发部张经理需要让各项目负责人了解代码安全保障系统中检测到的代码问题，并向各项目负责人发送检测结果的通知邮件，要求运维工程师小王对代码安全保障系统的邮箱进行配置。小王需要了解代码安全保障系统的授权状态，并配置邮箱服务器，请帮助小王对代码安全保障系统的系统配置进行设置。

【实验原理】

系统配置包括邮箱配置和授权信息。

发送报告之前需要先配置邮箱服务器，选择“系统管理”→“系统配置”→“邮箱配置”命令，输入服务器协议地址、发件人邮箱名和密码。在“报告管理”中单击“发邮件”按钮，输入收件人地址，收件人可收到报告下载地址的邮件。

授权信息显示的内容主要有以下几部分：

① 单位名称：展示 Key 中写入的信息。

② 管理平台版本号：展示平台配置文件写入的版本号。

③ 检测类型：显示 Key 中注册的检测类型并选中。

④ 检测语言：显示 Key 中注册的语言并选中。

⑤ 缺陷检测、合规检测和溯源检测：Key 中有该语言的注册信息和类型就显示，没有则不显示。

⑥ 检测引擎并发数：每个引擎可同时执行的最大任务数。

⑦ 检测引擎代理数：可配置的最大代理个数。

⑧ 最大注册用户数：系统中可注册的最大用户数。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 1-1 所示。

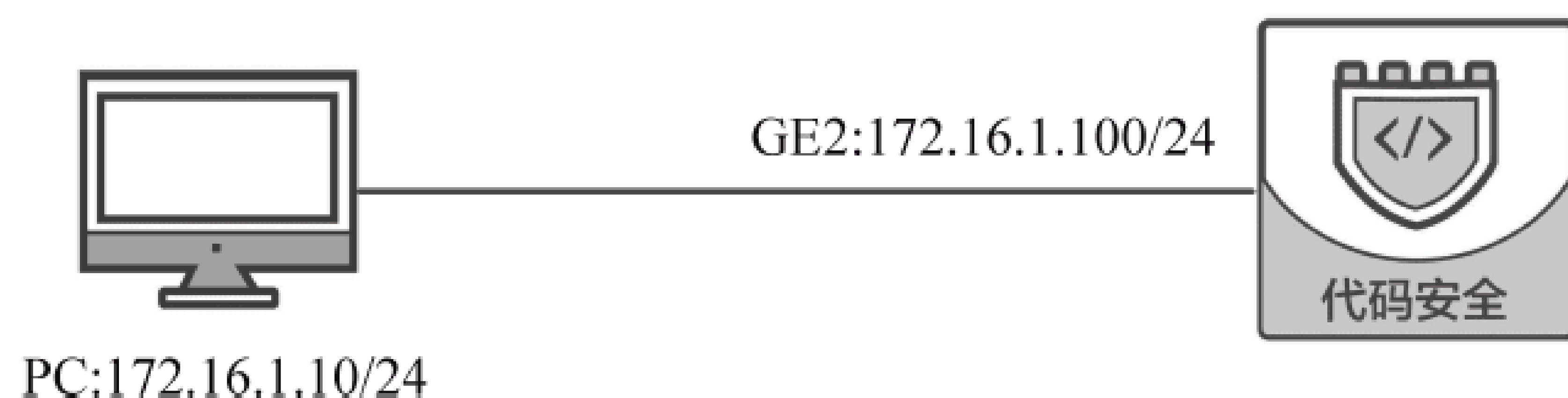


图 1-1　代码安全保障系统配置实验拓扑图

【实验思路】

(1) 配置代码卫士邮箱地址。

(2) 查看代码卫士系统的授权信息。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮，如图 1-2 所示。

(3) 单击“继续前往 172.16.1.100(不安全)” 按钮，如图 1-3 所示。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin，密码为“admin123!@＃”)，单击“登录”按钮，如图 1-4 所示。



图 1-2　单击“高级”按钮

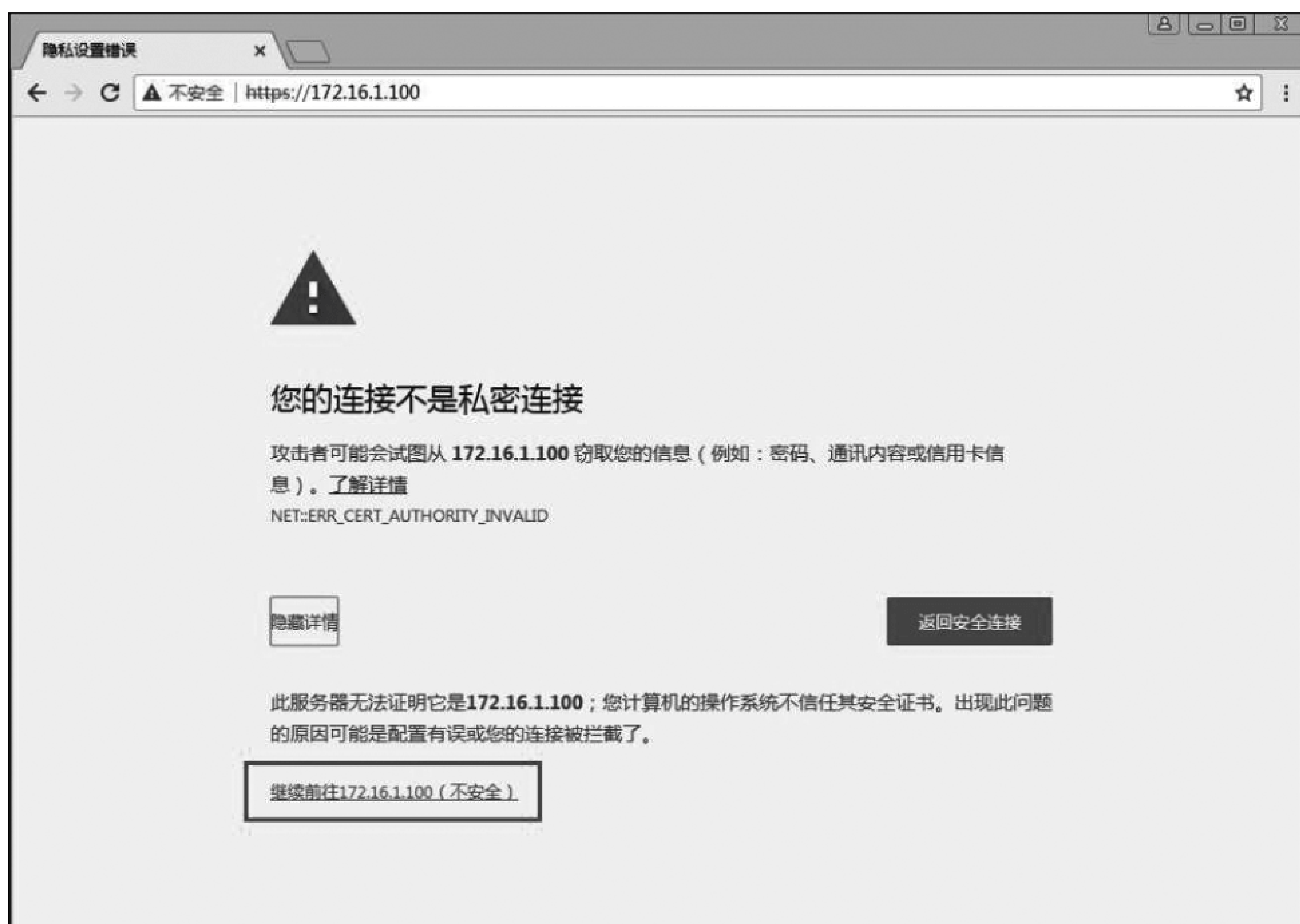


图 1-3　单击“继续前往 172.16.1.100(不安全)”按钮

(5) 登录成功后，选择“系统管理”命令，如图 1-5 所示。

(6) 选择左侧的“系统配置”命令，系统配置包括两部分：邮箱配置和授权信息，如图 1-6 所示。

(7) 选择“邮箱配置”命令，输入“邮箱服务器地址”、发送者的“邮箱名”及“密码”，单



图 1-4　登录代码卫士



图 1-5　系统管理



图 1-6　系统配置

击“提交”按钮即可配置好邮箱，如图 1-7 所示。



图 1-7　邮箱配置

（8）选择“授权信息”命令，查看代码卫士的授权信息，如图 1-8 所示。



图 1-8　查看授权信息

（9）在授权信息界面中，可以查看“单位名称”“管理平台版本号”“检测类型”“检测语言”，系统支持的“缺陷检测”“合规检测”和“溯源检测”的语言，以及“检测引擎并发数”和“最大引擎代理数”信息，如图 1-9、图 1-10 所示。

【实验预期】

代码卫士已经成功配置好发件人邮箱。



图 1-9　授权信息

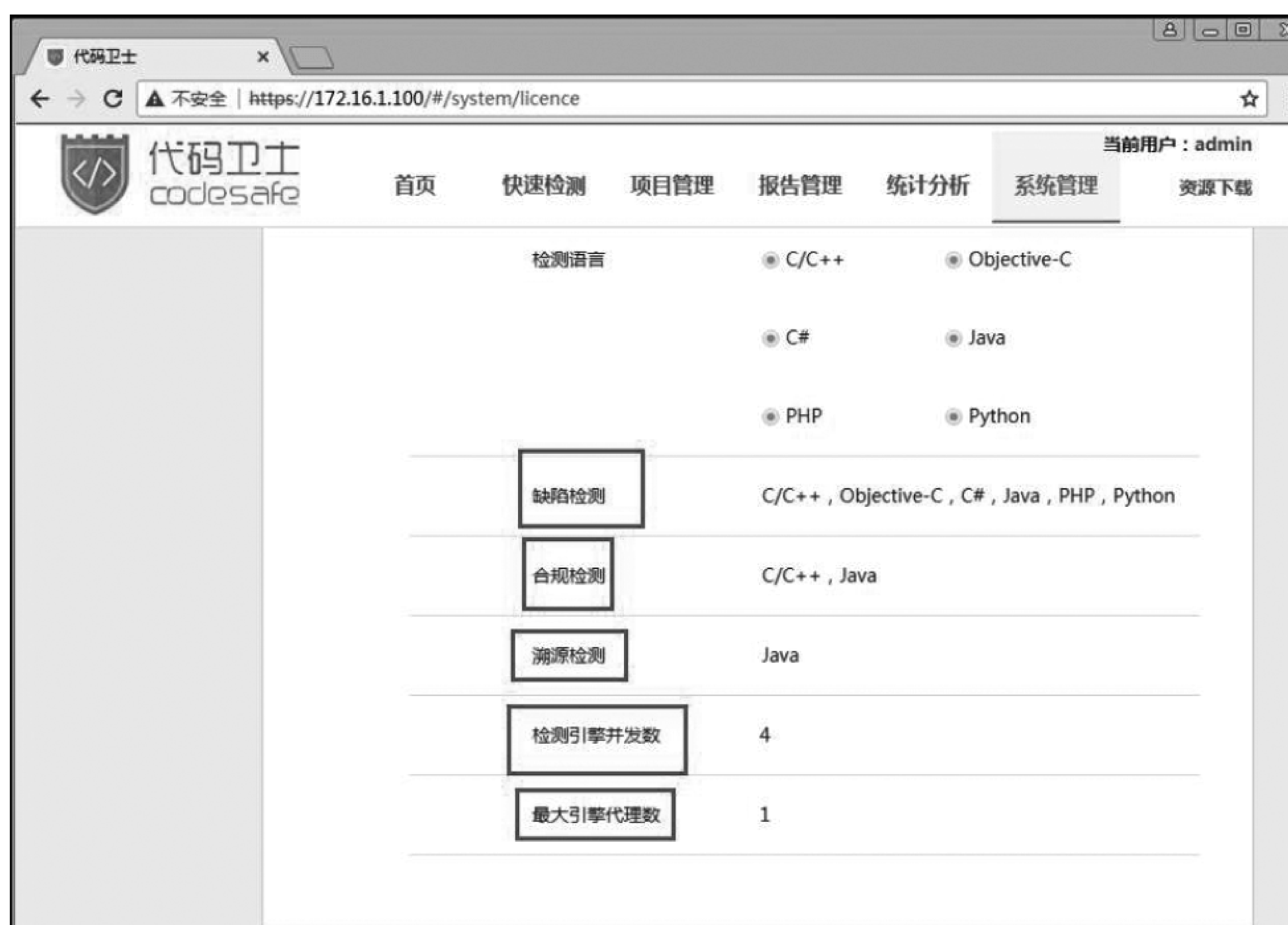


图 1-10　其他授权信息

【实验结果】

选择“邮箱配置”命令，发现系统已经成功配置好发件人邮箱，如图 1-11 所示。



图 1-11　邮箱配置

【实验思考】

如何将检测结果发送给项目负责人？

1.2　模块管理

1.2.1　代码安全保障系统用户管理实验

【实验目的】

管理员可以熟练掌握代码卫士系统的用户管理部分，包括新建部门、用户以及分配权限等。

【知识点】

用户管理。

【场景描述】

A 公司为提高交付软件代码质量购入了代码安全保障系统，张经理要求运维工程师小王针对公司的技术部、质管部、研发部进行权限划分，要求技术部负责分配账号、制定权限，研发部负责上传代码并检测，质管部负责检测结果汇总。小王需要对代码安全保障系统中的用户进行配置实现相关要求，请帮助小王实现用户管理。

【实验原理】

用户管理包括三部分：部门管理、用户管理和角色管理。

- 部门管理：部门管理包括部门的新增、修改、查找和删除，新建用户时必须要指定该用户所属部门。系统自带一个名为“系统默认”的部门。
- 用户管理：用户管理包括用户的新增、修改、查找、冻结和删除。系统自带用户名为 admin 的系统管理员用户。
- 角色管理：角色管理包括角色的新增、修改、查找和删除。系统自带系统管理员和用户两种角色。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 1-12 所示。

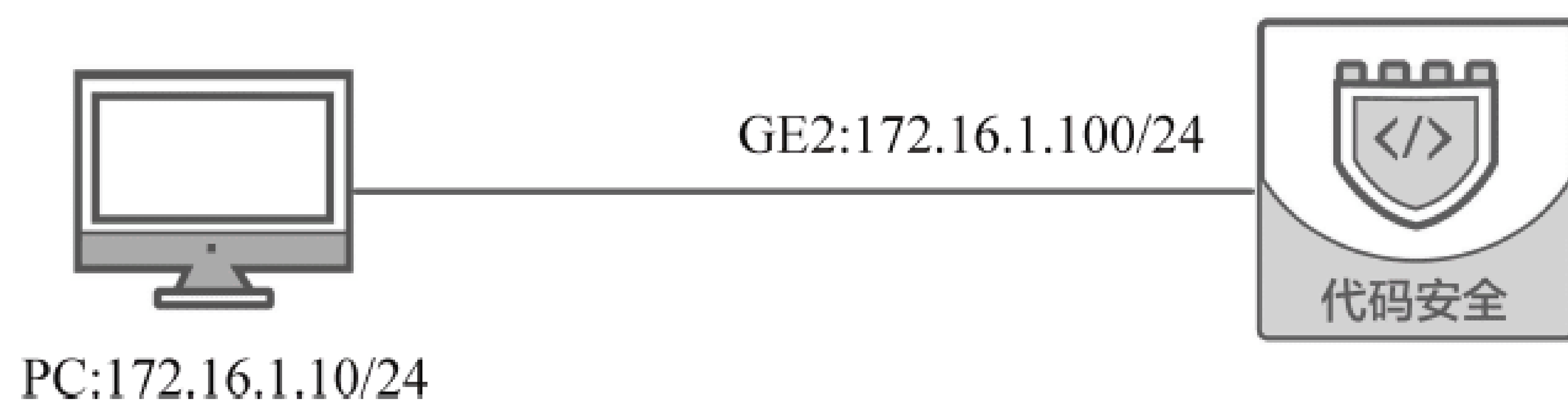


图 1-12　代码安全保障系统用户管理实验拓扑图

【实验思路】

添加部门、用户和角色。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(3) 单击“继续前往 172.16.1.100(不安全)”按钮。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(5) 登录成功后，选择“系统管理”命令。

(6) 选择“用户管理”→“部门”→“＋添加部门”命令，如图 1-13 所示。

(7) 在“添加部门”界面中，“部门名称”输入“技术部”，“描述”输入“负责分配账号，制定权限”，单击“保存”按钮，如图 1-14 所示。

(8) 单击“保存”按钮后，即可查看新添加的部门。单击“＋添加部门”按钮，如图 1-15 所示。

(9) 在“添加部门”界面中，“部门名称”输入“研发部”，“描述”输入“上传代码并检测”，



图 1-13　添加部门

添加部门

部门名称: 技术部

上级部门:

描述: 负责分配账号,制定权限

保存　关闭

图 1-14　添加技术部



图 1-15　再次添加部门

单击“保存”按钮。

(10) 单击“保存”后即可查看新添加的研发部。单击“+添加部门”按钮。

(11) 在“添加部门”界面中,“部门名称”输入“质管部”,“描述”输入“检测结果汇总”。

(12) 单击“保存”按钮后即可查看新添加的质管部。选择“角色”命令,如图 1-16 所示。



图 1-16　选择“角色”命令

(13) 代码卫士系统默认含有两个角色,用户和管理员。单击“+添加角色”按钮。

(14) 在“添加角色”界面中,“角色名称”输入“分配账号,制定权限”,“权限”选择“系统管理”下面的“用户管理”,单击“保存”按钮,如图 1-17 所示。



图 1-17　分配账号制定权限

（15）保存后即可查看新添加的角色。单击“＋添加角色”按钮，如图 1-18 所示。



图 1-18　添加角色

（16）在“添加角色”界面中，“角色名称”输入“上传代码并检测”，“权限”选择“快速检测”。

（17）保存后即可查看新添加的角色。单击“＋添加角色”按钮。

（18）在“添加角色”界面中，“角色名称”输入“检测结果汇总”，“权限”选择“统计分析”，单击“保存”按钮。

（19）保存后即可查看新添加的角色。选择“用户”命令，添加新用户，如图 1-19 所示。



图 1-19　添加用户

(20) 代码卫士系统默认含有一个系统管理员用户 admin,单击“+添加用户”按钮,添加新的用户,如图 1-20 所示。



图 1-20　再次添加用户

(21) 在“添加用户”界面中,“用户名”输入 jishu1,“密码”“确认密码”输入“jishu123!@#”,“部门”设置为“技术部”,“角色”设置为“分配账号,制定权限”,单击“保存”按钮,如图 1-21 所示。

图 1-21　添加 jishu1 用户

(22) 保存后系统返回用户列表界面，单击“＋添加用户”按钮。

(23) 在“添加用户”界面中，“用户名”输入 yanfa1，“密码”“确认密码”输入“yanfa123!@＃”，“部门”设置为“研发部”，“角色”设置为“上传代码并检测”，单击“保存”按钮。

(24) 保存后系统返回用户列表界面。单击“＋添加用户”按钮。

(25) 在“添加用户”界面中，“用户名”输入 zhiguan1，“密码”“确认密码”输入“zhiguan123!@＃”，“部门”设置为“质管部”，“角色”设置为“检测结果汇总”，单击“保存”按钮。

(26) 保存后系统返回用户列表界面，查看新添加的用户，如图 1-22 所示。



图 1-22　查看新用户

(27) 单击右上角的 admin 按钮，在显示的列表中单击“退出”按钮，如图 1-23 所示。



图 1-23　退出代码卫士系统

【实验预期】

（1）jishu1 用户可以分配账号并制定权限。

（2）yanfa1 可以上传代码并检测。

（3）zhiguan1 可以汇总检测结果。

【实验结果】

1. jishu1 用户可以分配账号并制定权限

（1）在代码卫士登录界面中，输入用户名 jishu1，密码输入“jishu123!@＃”，单击“登录”按钮。

（2）进入代码卫士首页后，选择“系统管理”命令，可以看到左侧栏中有对部门、用户及角色的管理，说明 jishu1 用户可以分配账号并制定权限，如图 1-24 所示。



图 1-24　用户 jishu1 登录

（3）单击右上角的 jishu1 按钮，在显示的列表中单击“退出”按钮。

2. yanfa1 可以上传代码并检测

（1）在代码卫士登录界面中，输入用户名 yanfa1，密码输入“yanfa123!@＃”，单击“登录”按钮。

（2）yanfa1 用户登录成功后进入代码卫士首页，选择“快速检测”命令，可以看到界面中有“缺陷检测”“合规检测”和“溯源检测”以及“＋发起快速检测”，说明用户 yanfa1 可以上传代码并检测，如图 1-25 所示。

（3）单击 yanfa1 按钮，在显示的列表中单击“退出”按钮。

3. zhiguan1 可以对检测结果进行汇总

（1）在代码卫士登录界面中，输入用户名 zhiguan1，密码输入“zhiguan123!@＃”，单击“登录”按钮。

（2）进入代码卫士首页，选择“统计分析”命令，可以看到左侧栏中有对“缺陷检测”“合规检测”和“溯源检测”的结果统计，说明用户 zhiguan1 可以对检测结果进行汇总，如图 1-26 所示。



图1-25　快速检测(1.2.1)



图1-26　统计分析

【实验思考】

(1) 假如你是技术部人员,尝试为研发部分配账号。

(2) 假如你是研发部人员,尝试将代码进行上传并检测。

1.2.2　代码安全保障系统检测模板管理实验

【实验目的】

通过本实验使管理员学会代码卫士的检测模板管理以及添加自定义检测模板。

【知识点】

检测模板管理。

【场景描述】

A公司同时进行多种开发语言项目的研发,为提高代码安全保障系统运行效率,张

经理要求运维工程师小王针对不同项目指定不同的检测模板,以提高检测效率。小王需要对代码安全保障系统的检测模板进行配置,请帮助小王实现检测模板的管理功能。

【实验原理】

检测模板分为缺陷检测模板和合规检测模板,系统对这两种检测方式都自带默认的模板,即勾选全部的缺陷/违规分类。

用户可以使用系统自带的检测模板,也可以根据实际情况,自定义检测模板,即自定义检测任务检测的缺陷/合规分类。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 1-27 所示。

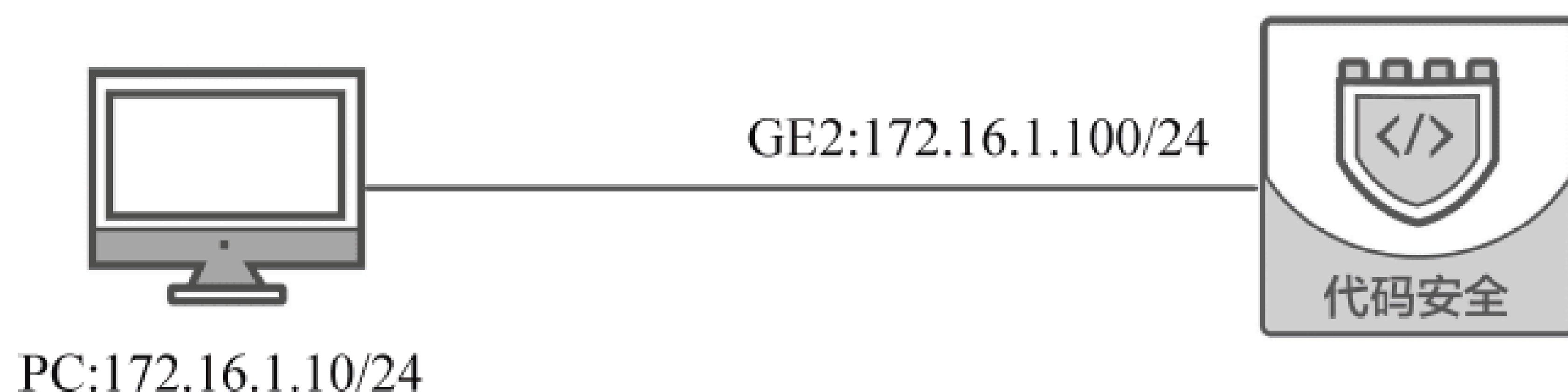


图 1-27　代码安全保障系统检测模板管理实验拓扑图

【实验思路】

(1) 查看系统默认检测模板。

(2) 自定义一个检测模板。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开 Chrome 浏览器,输入代码卫士的 IP 地址"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(3) 单击"继续前往 172.16.1.100(不安全)"按钮。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(5) 登录成功后,选择"系统管理"命令。

(6) 选择"检测模板管理"命令,可以看到系统具有缺陷检测和合规检测两种功能,如图 1-28 所示。

(7) 选择"缺陷检测"命令,发现系统内置了 C/C++、Objective-C、C#、Java、PHP、Python 六种开发语言的检测模板,如图 1-29 所示。

(8) 选择"查看"命令,查看"缺陷检测模板[C/C++]"的模板信息,如图 1-30 所示。

(9) 在"缺陷检测模板[C/C++]"中,管理员可以指定此模板的访问权限、检测模板适



图 1-28　检测模板管理



图 1-29　缺陷检测模板



图 1-30　查看模板信息

用的开发语言以及模板的缺陷分类，缺陷分类可以选择其中某一种或某几种，也可以全部选中，查看完毕后单击“关闭”按钮，如图 1-31 所示。

（10）选择“合规检测”命令，如图 1-32 所示。



图 1-31　查看缺陷信息



图 1-32　合规检测

(11) 代码卫士默认配置了 C/C++和 Java 语言的缺陷检测，选择“合规检测模板[C/C++]”按钮右侧的“查看”命令，如图 1-33 所示。

(12) 在模板名称为“合规检测模板[C/C++]”的模板中，可以指定“访问权限”“开发语言”和“违规分类”，查看完毕后单击“关闭”按钮，如图 1-34 所示。

(13) 自定义缺陷检测模板。选择“缺陷检测”→“+添加模板”命令，添加一个缺陷检测模板，如图 1-35 所示。

(14) 在“添加模板”界面中，“模板名称”输入“缺陷检测模板测试”，“访问权限”选中



图 1-33　选择“查看”命令



图 1-34　单击“关闭”按钮



图 1-35　添加缺陷检测模板

"仅自己"单选钮,"开发语言"设置为 PHP,"缺陷分类"设置为全选,"描述"输入"模板测试",单击"保存"按钮,如图 1-36 所示。

图 1-36　继续添加缺陷检测模板

【实验预期】

缺陷检测模板中新增一个模板,且新建代码检测时可以选择此模板。

【实验结果】

(1) 单击"保存"按钮后,缺陷检测模板列表中新添加了一个名为"缺陷检测模板测试"的模板,如图 1-37 所示。



图 1-37　新增一个模板

（2）选择“快速检测”→“＋发起快速检测”命令，如图1-38所示。



图1-38　快速检测(1.2.2)

（3）“开发语言”选中PHP单选钮，在“缺陷检测”选项中单击下拉框，可以看到新添加的缺陷检测模板，说明新添加的缺陷检测模板可以使用，如图1-39所示。



图1-39　缺陷模板可以使用

【实验思考】

（1）自定义一个针对Java合规检测的检测模板。

（2）尝试使用新添加的检测模板发起一个快速检测。

1.2.3 代码安全保障系统日志管理实验

【实验目的】

通过本实验使管理员熟练掌握查看并分析代码卫士的操作日志和异常日志。

【知识点】

日志管理。

【场景描述】

A 公司的运维部门需要汇总代码安全保障系统的日志内容便于项目管理,运维工程师小王要获取日志中相关的操作日志和异常日志。请帮助小王获取代码安全保障系统的日志,并对日志内容进行分析。

【实验原理】

日志管理分为操作日志、异常日志和系统日志。操作日志记录用户对系统的操作,包括用户/部门/角色的新增、修改和删除;发起、修改、删除检测任务等和操作相关的日志;异常日志记录异常情况,如任务检测失败、异常操作等;系统日志罗列显示程序后台输出的日志,方便定位问题。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 1-40 所示。

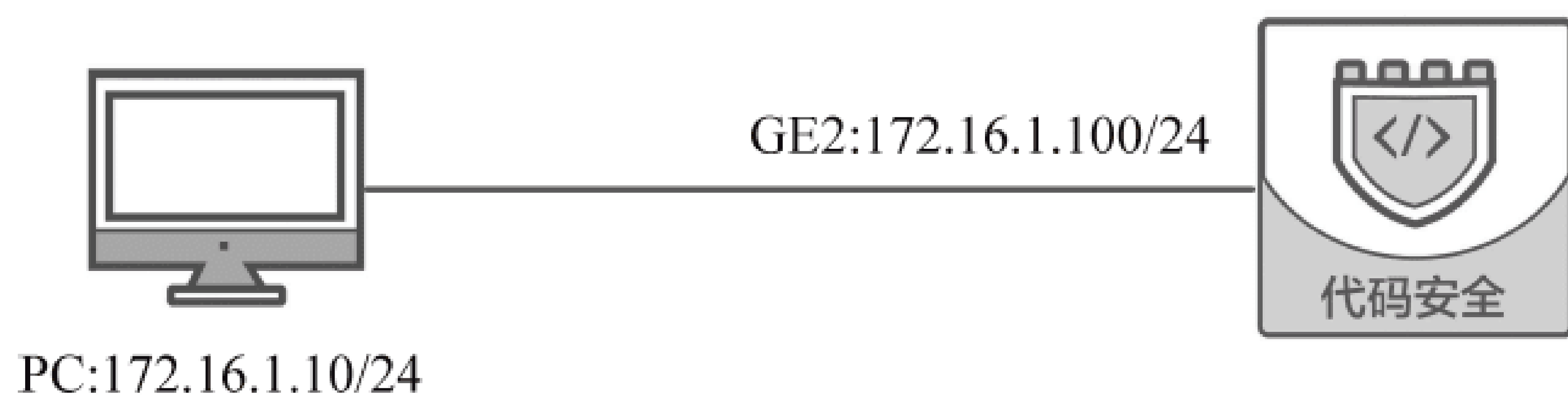


图 1-40 代码安全保障系统日志管理实验拓扑图

【实验思路】

(1)通过发起任务、添加用户产生日志。

(2)查看并分析操作日志和异常日志。

【实验步骤】

(1)进入实验平台对应的实验拓扑,登录左侧的 PC 主机终端虚拟机,如需登录密码,输入 123456。

(2)打开 Chrome 浏览器,输入代码卫士的 IP 地址“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(3) 单击“继续前往 172.16.1.100(不安全)”按钮。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(5) 登录成功后,选择“系统管理”命令。

(6) 选择“快速检测”→“+发起快速检测”命令。

(7) “任务名称”输入 test,“开发语言”选中 PHP 单选钮,“源码来源”选中“本地”单选钮,单击“上传文件”右侧的“浏览”按钮,选择桌面上的“WordPress-4.5.1.zip”文件并上传,其他保持默认配置,单击“发起检测”按钮,如图 1-41 所示。



图 1-41　发起检测

(8) 单击“+发起快速检测”按钮,如图 1-42 所示。



图 1-42　快速检测(1.2.3)

(9) “任务名称”输入 test,“开发语言”选中 PHP 单选钮,“源码来源”选中 Git 单选钮,“Git 地址”输入“http://192.168.1.1/git/git-test”,“Git 分支名称”输入 master,“Git 用户名”输入 git,“Git 密码”输入 123456,其他保持默认配置,单击“发起检测”按钮,如图

1-43 所示。

图 1-43　发起检测界面

(10) 发起检测后,系统回到任务列表界面。由于所给 Git 服务器地址不存在,因此任务检测失败。选择所有任务,单击“删除选中项”按钮,在弹出的窗口中单击“确定”按钮,如图 1-44 所示。

图 1-44　删除选中项

(11) 选择“系统管理”命令。

(12) 选择“用户管理”→“部门”→“＋添加部门”命令。

(13) 在“添加部门”界面中,“部门名称”输入 test,单击“保存”按钮。

(14) 选择“用户管理”→“用户”→“＋添加用户”命令。

(15) 在“添加用户”界面中,“用户名”输入 test,“密码”“确认密码”输入“test123!@＃”,“部门”设置为 test,“角色”设置为“用户”,单击“保存”按钮。

（16）选中添加的用户 test，选择“删除选中项”命令，在弹出的窗口中单击“确定”按钮，如图 1-45 所示。



图 1-45　删除用户

（17）选择“部门”命令，选中添加的部门 test，选择“删除选中项”命令，在弹出的窗口中单击“确定”按钮，如图 1-46 所示。



图 1-46　删除部门

【实验预期】

操作日志中记录用户对系统的所有操作。

【实验结果】

（1）选择“日志管理”→“操作日志”命令，如图 1-47 所示。



图 1-47　查看操作日志

(2) 系统记录了用户 admin 的登录信息,发起任务检测,删除任务,以及添加、删除用户和部门的所有操作信息,包括操作时间、操作人所用 IP、操作所属模块以及操作内容,如图 1-48 所示。

首页 快速检测 项目管理 报告管理 统计分析 系统管理 当前用户:admin 资源下载

操作日志列表 导出日志 搜索

操作时间	操作人IP地址	操作人	操作模块	操作内容
2018-03-18 20:54:39	172.16.1.10	admin	系统管理	删除部门:test
2018-03-18 20:54:39	172.16.1.10	admin	系统管理	删除用户:test
2018-03-18 20:53:21	172.16.1.10	admin	系统管理	添加用户:test
2018-03-18 20:50:42	172.16.1.10	admin	系统管理	添加部门:test
2018-03-18 20:45:40	172.16.1.10	admin	快速检测	删除了任务:(test)
2018-03-18 20:45:40	172.16.1.10	admin	快速检测	删除了任务:(test)
2018-03-18 20:43:53	172.16.1.10	admin	快速检测	发起了任务:(test)
2018-03-18 20:42:40	172.16.1.10	admin	快速检测	发起了任务:(test)
2018-03-18 20:41:54	172.16.1.10	admin	快速检测	删除了任务:(test)
2018-03-18 20:41:49	172.16.1.10	admin	快速检测	删除了任务:(test)
2018-03-18 20:39:20	172.16.1.10	admin	快速检测	发起了任务:(test)
2018-03-18 20:32:34	172.16.1.10	admin	快速检测	发起了任务:(test)
2018-03-18 20:31:31	172.16.1.10	admin	系统管理	admin登录系统
2018-03-18 20:01:10	172.16.1.10	admin	系统管理	添加了模板:缺陷检测模板测试,模板测...
2018-03-18 19:52:39	172.16.1.10	admin	系统管理	admin登录系统

删除选中项

图 1-48 操作日志

(3) 单击"异常日志"按钮,异常日志记录异常情况,如任务检测失败、异常操作等。图 1-49 显示的是之前发起 Git 检测任务失败时产生的异常日志。

代码卫士 codesafe 首页 快速检测 项目管理 报告管理 统计分析 系统管理 当前用户:admin 资源下载

用户管理 检测模板管理 日志管理 操作日志 异常日志 系统日志

异常日志列表 导出日志 搜索

发生时间	操作人IP地址	操作人	异常模块	异常描述
2018-03-18 20:43:53	172.16.1.100	服务器	调度中心	Sky引擎缺陷检测任务(test)检...

删除选中项

图 1-49 异常日志

【实验思考】

(1) 如何导出操作日志?

(2) 如何删除某些日志?

1.2.4 代码安全保障系统引擎管理实验

【实验目的】

通过本实验学会修改代码卫士最大引擎数,查看当前系统检测状态。

【知识点】

引擎管理。

【场景描述】

A 公司研发部项目经理老张需要了解当前代码安全保障系统中各项目开发、检测情况，需要通过代码安全保障系统中的引擎中的引擎查询和队列管理了解当前的检测状态。请帮助张经理对代码安全保障系统的引擎管理进行配置。

【实验原理】

引擎管理包括两部分：引擎查询和队列管理，用以提供引擎信息、检测队列的查看等功能。

- 引擎查询：提供引擎信息查看的功能，可以查看系统配置的引擎情况（包括操作系统类型、IP、CPU、内存和硬盘容量），展开列表可以查看每种语言检测引擎的版本号；并发数是指同时执行任务的个数，并发数为 1，表示同一时间只能执行一个任务（包含快速检测和项目管理），其他发起的任务只能在排队中，状态为“排队中”。
- 队列管理：查看所有排队中和检测中的任务。对排队中的任务可进行暂停、开始、置顶和删除操作，对检测中的任务只能删除。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 1-50 所示。

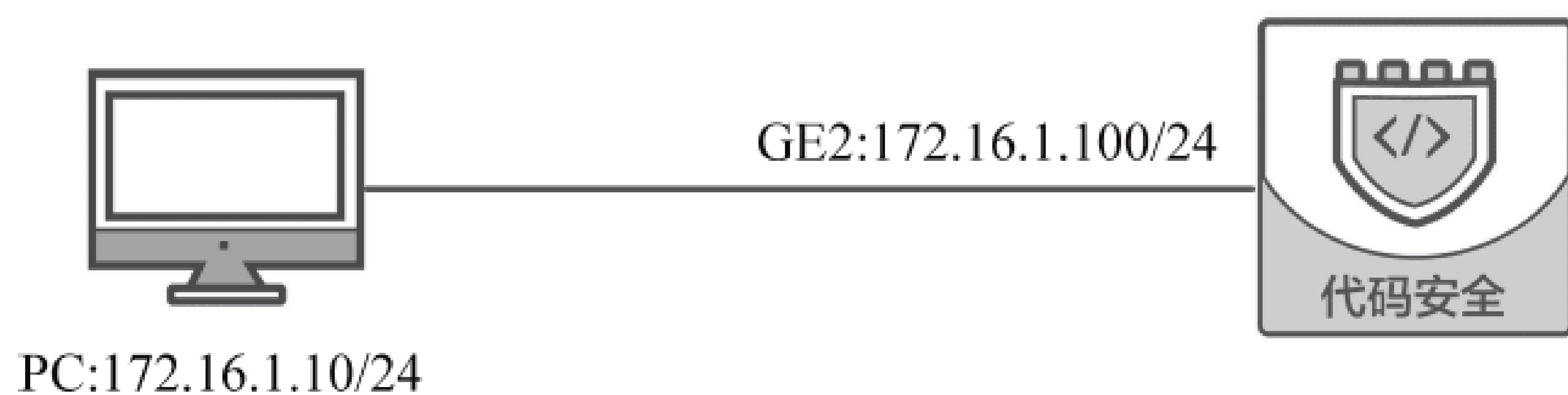


图 1-50　代码安全保障系统引擎管理实验拓扑图

【实验思路】

（1）介绍代码卫士引擎管理。

（2）发起检测任务，观察队列管理状态。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(3) 单击“继续前往 172.16.1.100(不安全)”按钮。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(5) 登录成功后,选择“系统管理”命令。

(6) 选择“引擎管理”命令,引擎管理包括两部分:引擎查询和队列管理,如图 1-51 所示。



图 1-51 引擎管理

(7) 选择“引擎查询”命令,可以看到引擎的 IP 地址、所用的操作系统类型、CPU 类型、内存、硬盘容量以及最大并发数,如图 1-52 所示。



图 1-52 引擎查询

(8) 单击 IP 命令下面的“+”按钮,查看系统所支持的检测语言以及检测类型,如图 1-53 所示。

(9) 单击“最大并发数”按钮下面的数字,可以修改最大并发数,将最大并发数改为 1。

IP	操作系统	CPU	内存	硬盘容量	最大并
172.16.1.101	Windows	CPU E5-2630	128G	1T	1

检测类型	检测工具	语言	版本
缺陷检测	CodeSafe	C/C++	1.7.0
合规检测	CodeSafe	C/C++	1.7.0
缺陷检测	CodeSafe	Objective-C	1.7.0
缺陷检测	CodeSafe	C#	1.4.2
缺陷检测	CodeSafe	Java	3.1.5
合规检测	CodeSafe	Java	3.1.5
溯源检测	CodeSafe	Java	1.0.2
缺陷检测	CodeSafe	PHP	2.1.14
缺陷检测	CodeSafe	Python	2.0.3

图 1-53　继续引擎查询

最大并发数是指引擎系统支持的最大同时检测数量，如图 1-54 所示。

IP	操作系统	CPU	内存	硬盘容量	最大并发数
172.16.1.101	Windows	CPU E5-2630	128G	1T	1

检测类型	检测工具	语言	版本
缺陷检测	CodeSafe	C/C++	1.7.0
合规检测	CodeSafe	C/C++	1.7.0
缺陷检测	CodeSafe	Objective-C	1.7.0
缺陷检测	CodeSafe	C#	1.4.2
缺陷检测	CodeSafe	Java	3.1.5
合规检测	CodeSafe	Java	3.1.5
溯源检测	CodeSafe	Java	1.0.2
缺陷检测	CodeSafe	PHP	2.1.14
缺陷检测	CodeSafe	Python	2.0.3

图 1-54　最大并发数

（10）选择“队列管理”命令，队列管理中会显示当前正在检测的任务以及处于排队状态的任务，如图 1-55 所示。



图 1-55　队列管理

【实验预期】

最大并发数是 1 时，连续发起两个检测任务，第二个处于排队状态。

【实验结果】

(1) 选择“快速检测”→“＋发起快速检测”命令。

(2) “任务名称”输入 test1，“开发语言”选中 PHP 单选钮，“源码来源”选中“本地”单选钮，单击“上传文件”右侧的“浏览”按钮，选择桌面上的“WordPress-4.5.1.zip”并上传，其他保持默认配置，单击“发起检测”按钮。

(3) 重复步骤(1)(2)，发起第二个检测，可以看到由于最大并发数是 1，所以发起的第二个检测处于排队中，如图 1-56 所示。



图 1-56　排队中

(4) 选择“系统管理”→“引擎管理”→“队列管理”命令，查看处于排队中的任务，如图 1-57 所示。



图 1-57　查看处于排队中的任务

【实验思考】

将最大并发数改成 2，重复上述步骤，观察队列变化。

第 2 章 代码安全保障系统缺陷检测

代码安全保障系统支持 Windows、Linux 等多种操作系统平台上软件源代码的缺陷检测。支持 C/C++/C#/Java/JSP/JavaScript/Python/HTML/XML/Cobol 等主流编程语言。可检测的缺陷种类包括缓冲区溢出、SQL 注入、跨站脚本、代码质量、危险函数等 13 个大类,600 多个小类。

本章主要完成代码安全保障系统缺陷检测实验,根据主流编程语言将本章划分为 3 个模块:C/C++缺陷检测、PHP 缺陷检测和 Java 缺陷检测。通过完成这些实验,检查源代码中存在的安全缺陷,了解应用程序中最可能、最常见、最危险的安全隐患,代码安全保障系统针对这些隐患,提供相应的消除建议,帮助程序员编写更安全的代码,并帮助管理员衡量软件的安全性。

2.1 C/C++缺陷检测

2.1.1 代码释放后使用缺陷检测实验

【实验目的】

确保所编写的代码中不存在使用已经释放后的内存的情况。

【知识点】

"释放后使用"缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个内存处理模块,调用 malloc/free 库函数为指针反复申请/释放内存,需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:释放后使用。

【实验原理】

释放后重用漏洞是一类典型的软件漏洞,它的成因是编程人员对内存管理疏忽,使用了释放后的内存块,这种行为是非常危险的,释放后的内存块包含的数据轻则是一些脏数据,导致程序不能完成指定功能,严重的情况下可能是攻击者构造的攻击程序,执行了这

段程序可能被恶意攻击。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-1 所示。

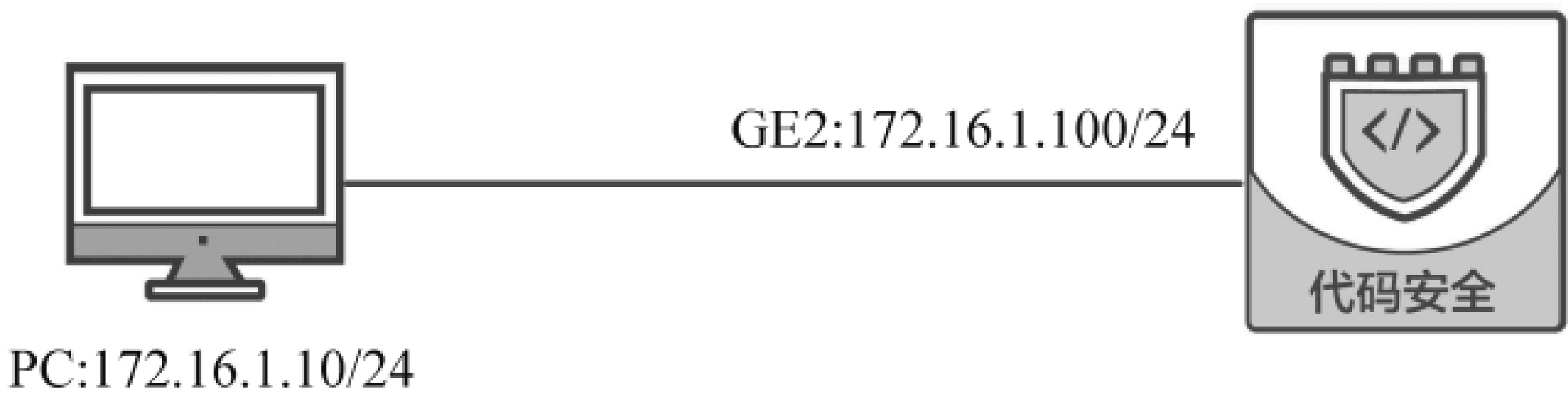


图 2-1　代码释放后使用缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面，如图 2-2 所示。

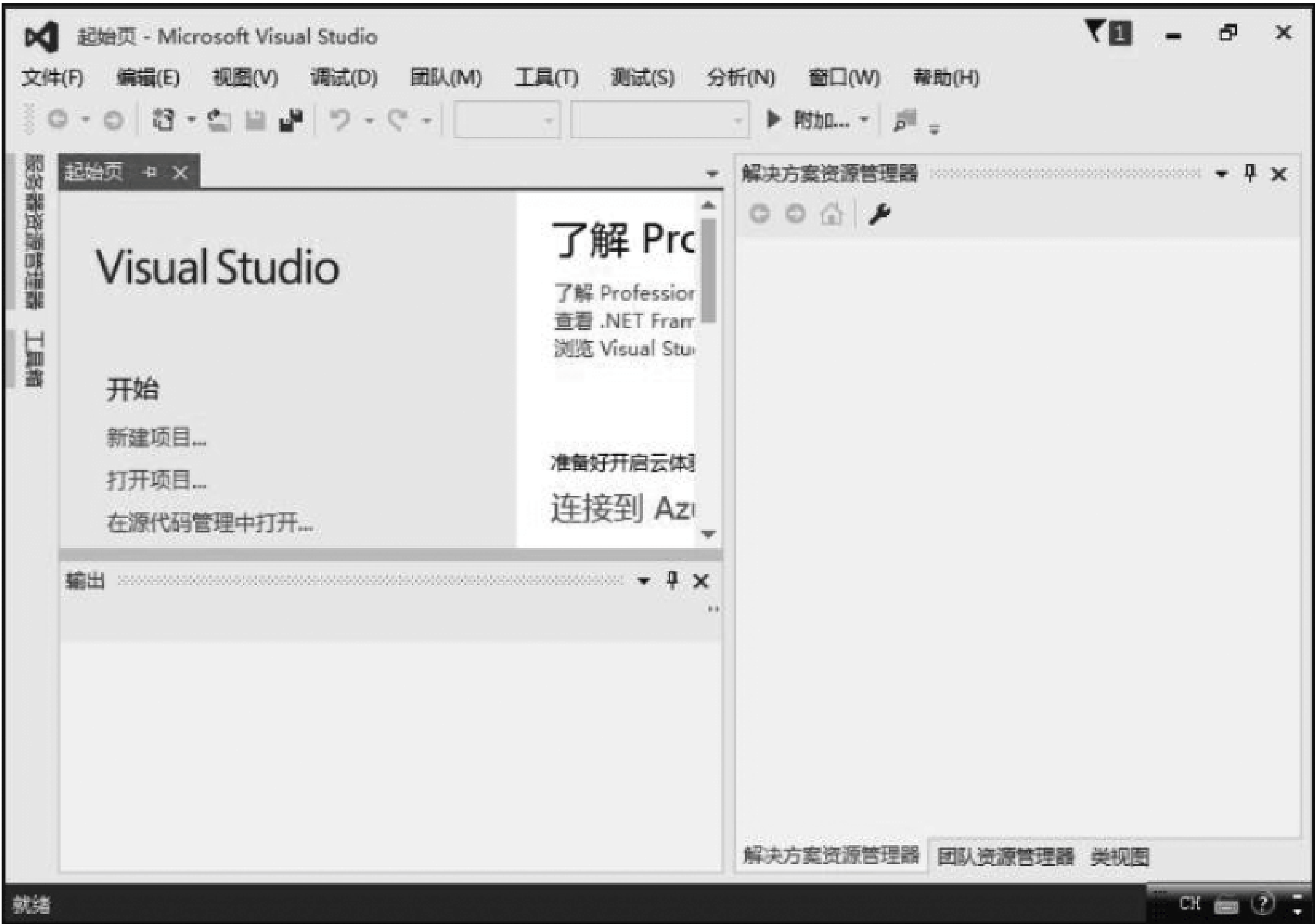


图 2-2　Visual Studio 2015 主界面

(3) 进入主界面,选择“文件”→“新建”→“项目”命令,新建一个项目,如图 2-3 所示。



图 2-3 新建一个项目

(4) 进入“新建项目”界面,在左侧“模板”选择“Visual C++”命令,界面中间选项选择“空项目”,下侧“名称”选项框内输入 code,单击“位置”右侧的“浏览”按钮,选择“C:\”,其他默认选择。单击“确定”按钮,如图 2-4 所示。

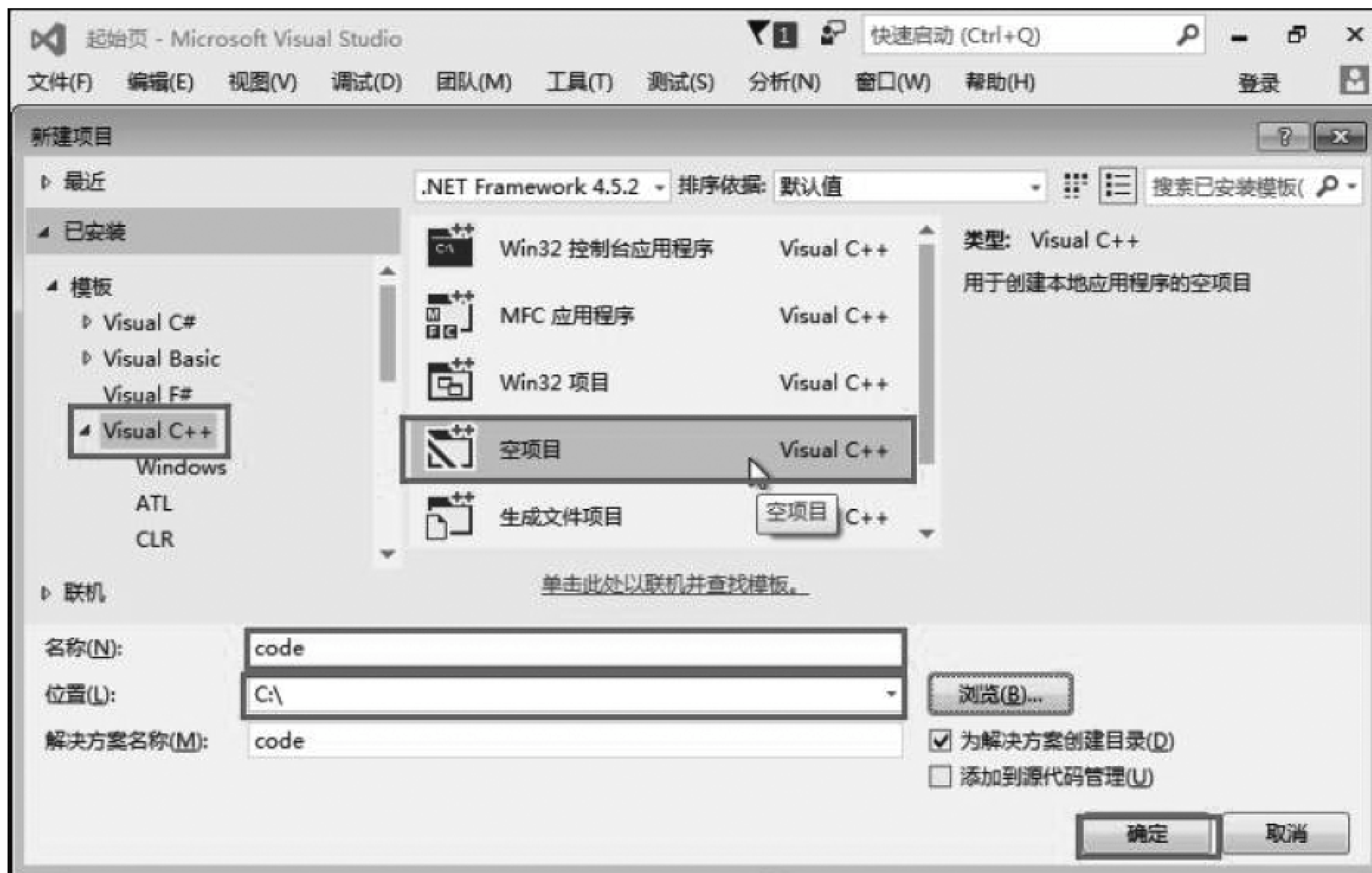


图 2-4 新建项目

（5）再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项，如图 2-5 所示。

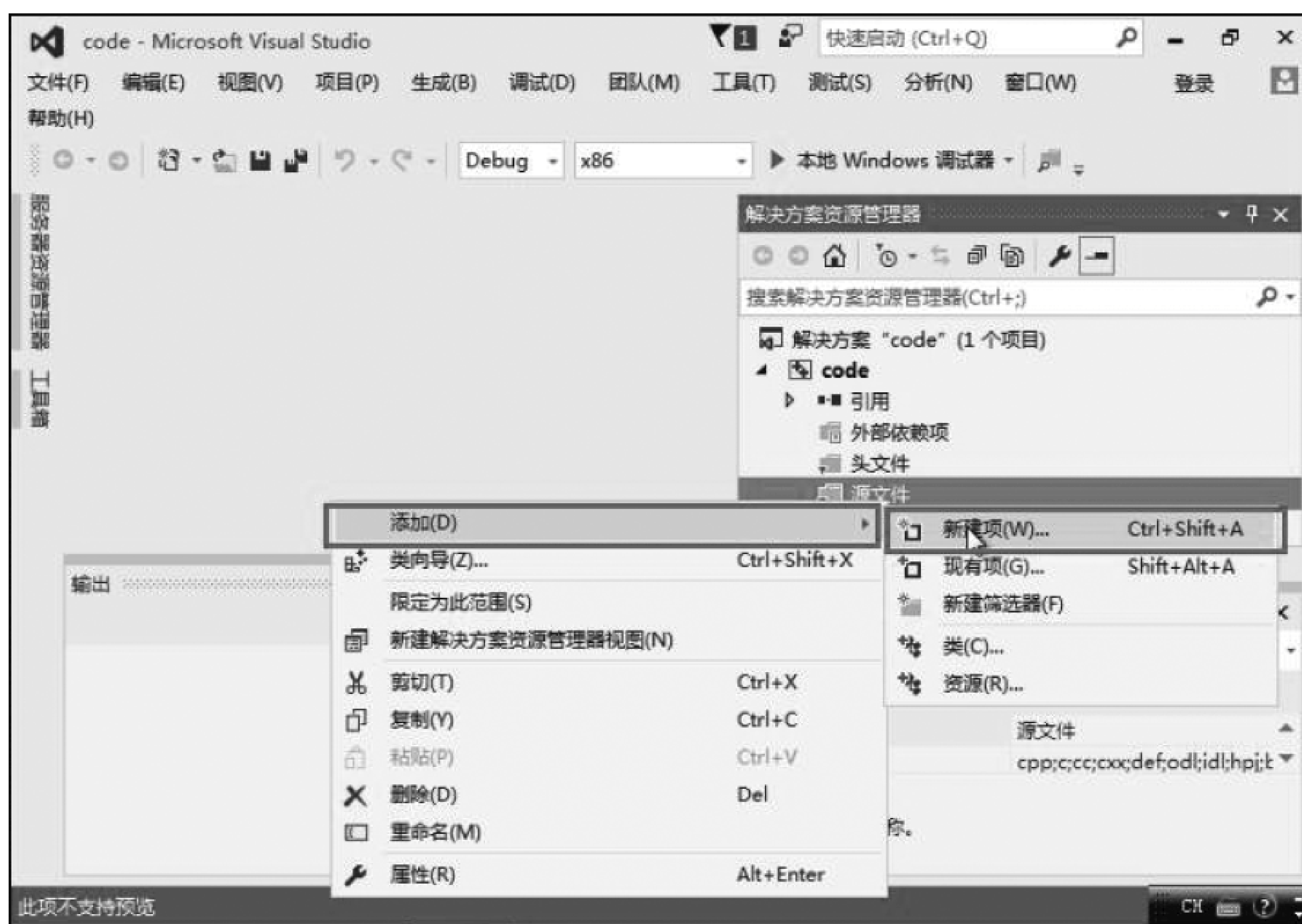


图 2-5 新建项

（6）进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，界面中间选项选择“C++文件(.cpp)”，在下侧“名称”选项框内输入 code，单击“添加”按钮，如图 2-6 所示。



图 2-6 “添加新项”界面

(7) 打开“C:\”下的 code.txt,复制其中的代码,如图 2-7 所示。



图 2-7　复制代码(2.1.1)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,单击“本地 Windows 调试器”按钮,如图 2-8 所示。

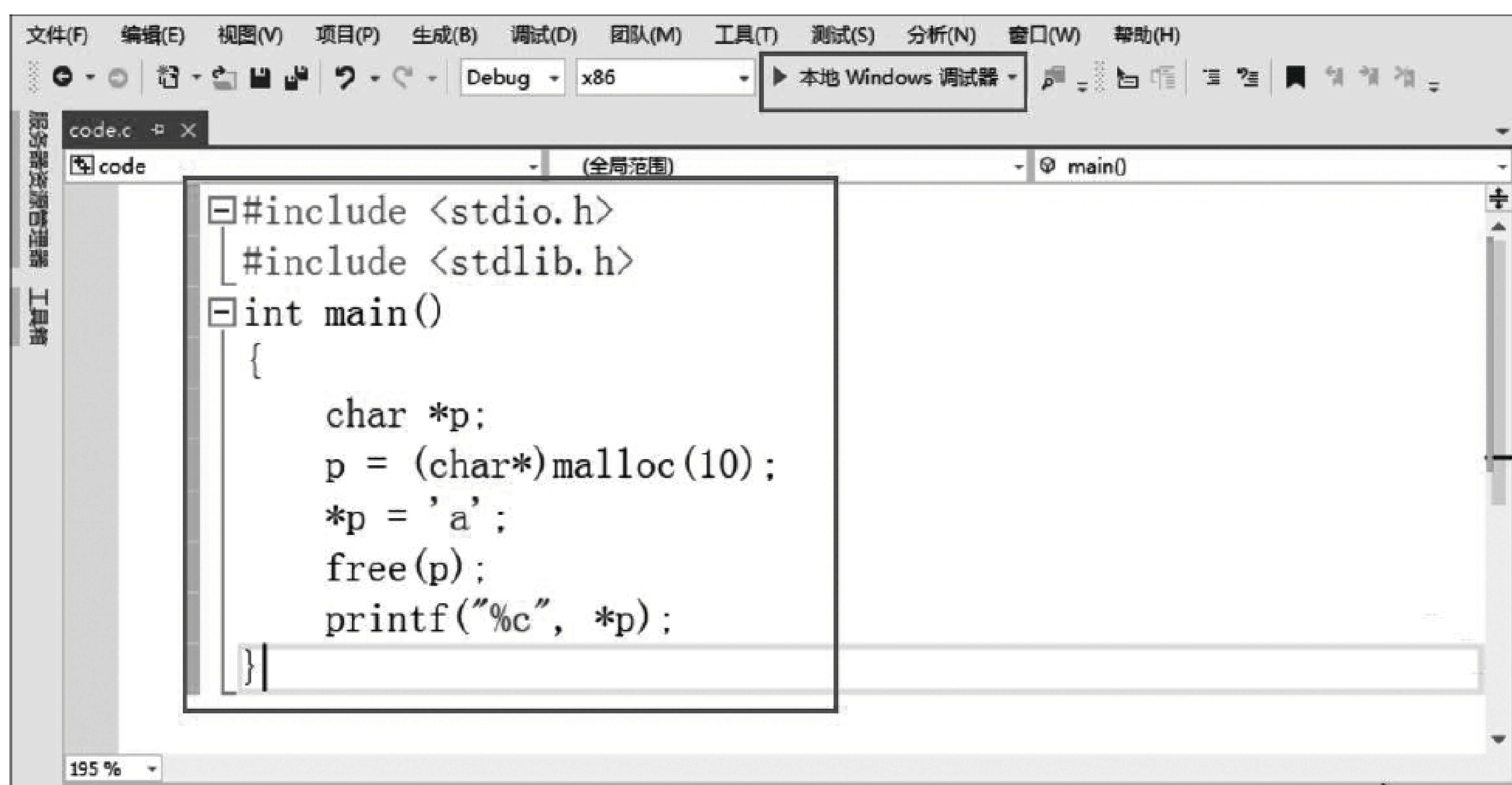


图 2-8　代码编辑界面

(9) 弹出“编译确认”对话框,单击“是”按钮,如图 2-9 所示。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面,如图 2-10 所示。

(11) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”,其中,“C:\codemanager”为中间表示文件的生成路径(生成路径以实际为准),“C:\code\code.sln”为刚刚编写的项目代码的存储路径(存储路径以实际设置的存储路径为准)。codemanager 文件夹会通过此行命令生成,如果“C:\”目录下已经生成此文件夹,必须清



图 2-9　编译确认



图 2-10　打开“VS 2015 开发人员命令提示”界面

空后再键入此行命令。按 Enter 键执行，显示执行成功提示，生成中间表示文件，如图 2-11 所示。

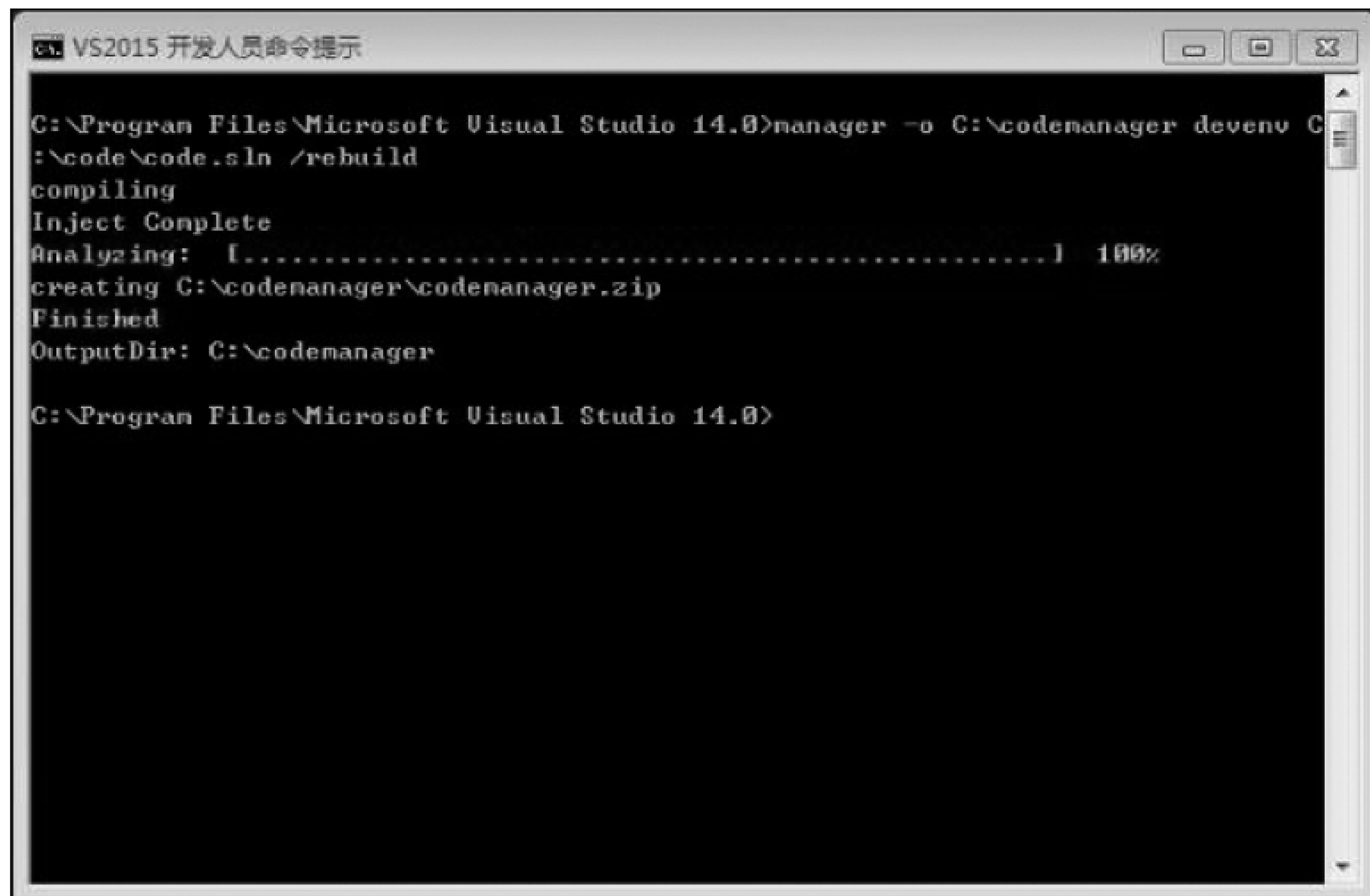


图 2-11　中间表示文件生成

（12）进入终端机“C:\”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功，如图 2-12 所示。

（13）打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。



图 2-12　查看中间表示文件

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”输入“释放后使用”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框,如图 2-13 所示。



图 2-13　发起快速检测

(18) 单击“浏览”按钮,如图 2-14 所示。

(19) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮,如图 2-15 所示。

(20) 选择 codemanager,单击“打开”按钮,如图 2-16 所示。

(21) 返回“快速检测”界面,等待文件上传成功,单击“发起检测”按钮,如图 2-17 所示。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。



图 2-14　上传代码

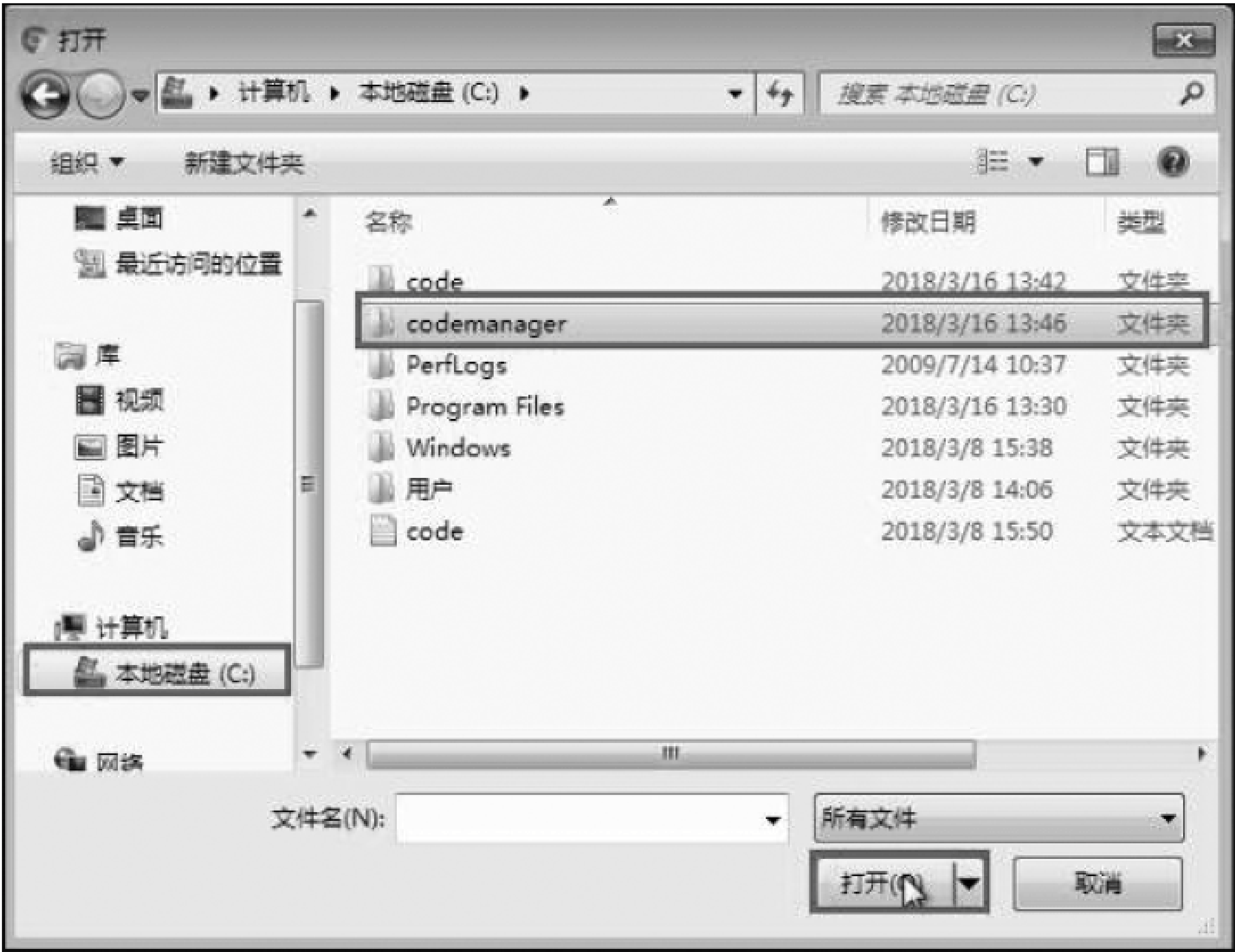


图 2-15　继续上传代码

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

（1）在学生本地机打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@#”，单击“登录”按钮。

（2）单击“+发起快速检测”按钮，查看列表所示检测结果，如图 2-18 所示。

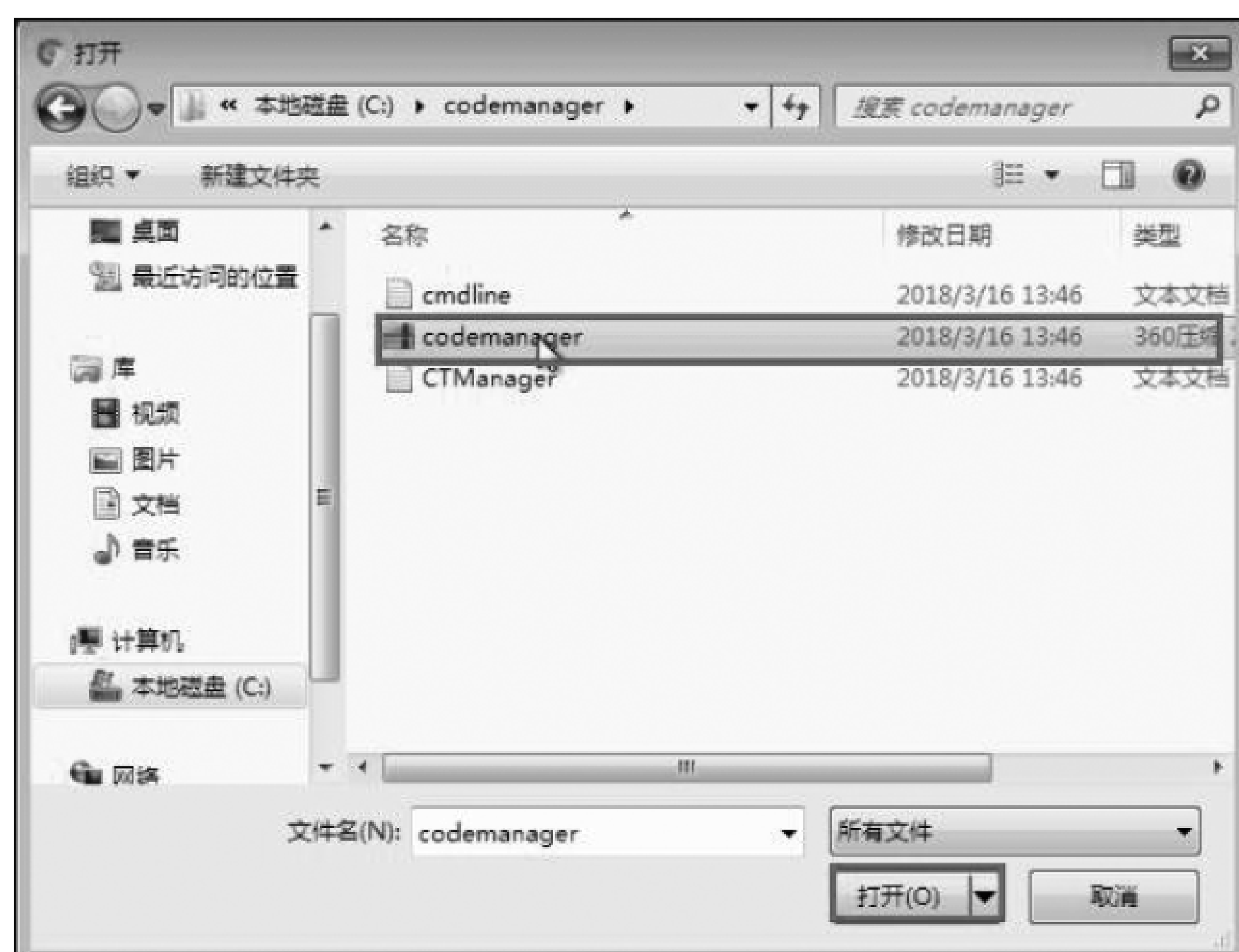


图 2-16　单击“打开”按钮



图 2-17　发起检测



图 2-18　查看列表所示检查结果

(3) 找到“释放后使用”,单击右侧的“缺陷审计”按钮,如图 2-19 所示。



图 2-19　“快速检测”界面(2.1.1)

(4) 在左侧查看代码的缺陷,如图 2-20 所示。



图 2-20　查看代码缺陷

(5) 打开“C:\code”文件夹,选择 code.sln,打开项目,如图 2-21 所示。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015,如图 2-22 所示。

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件,如图 2-23 所示。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令

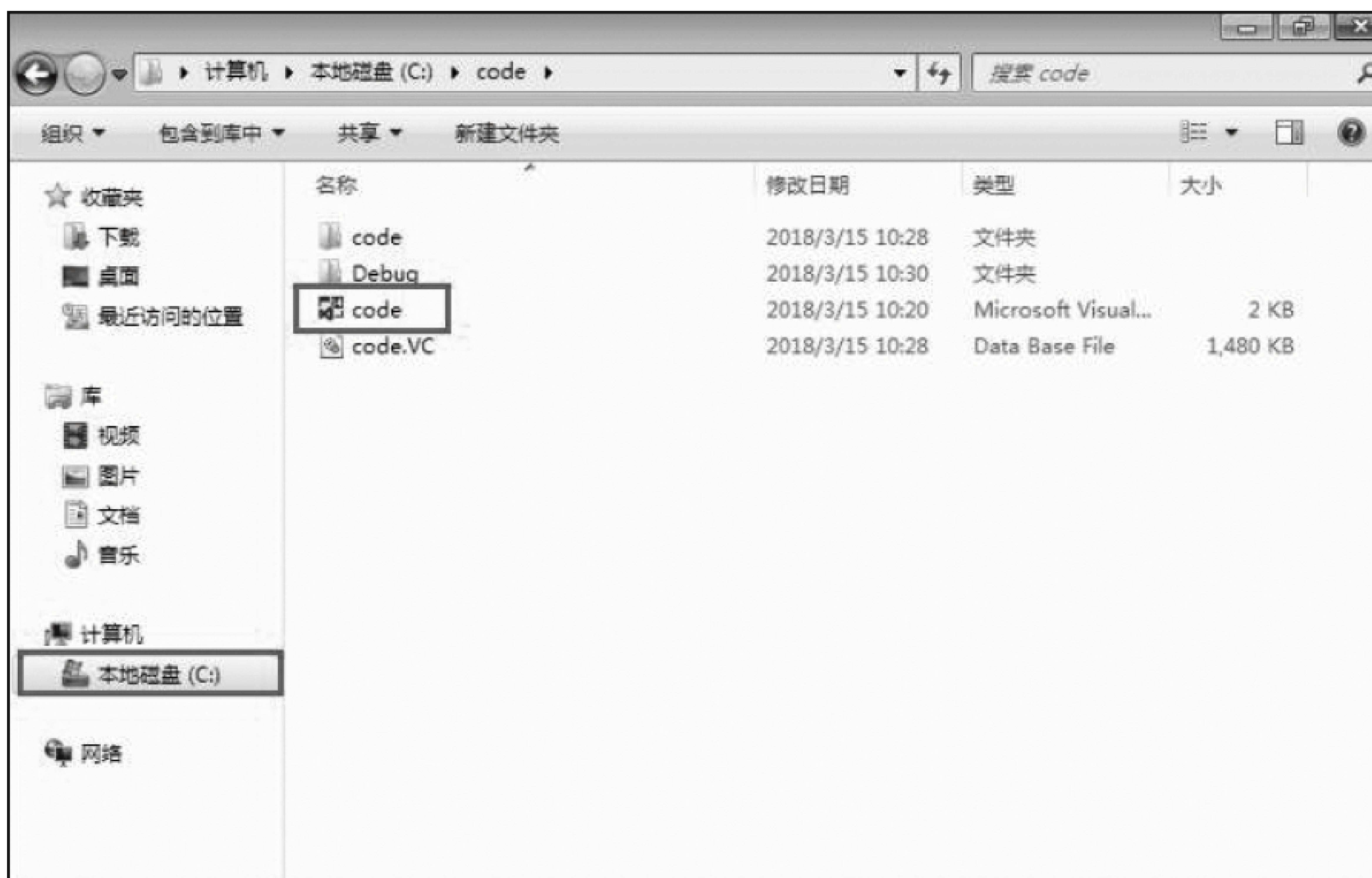


图 2-21　code.sln 文件位置

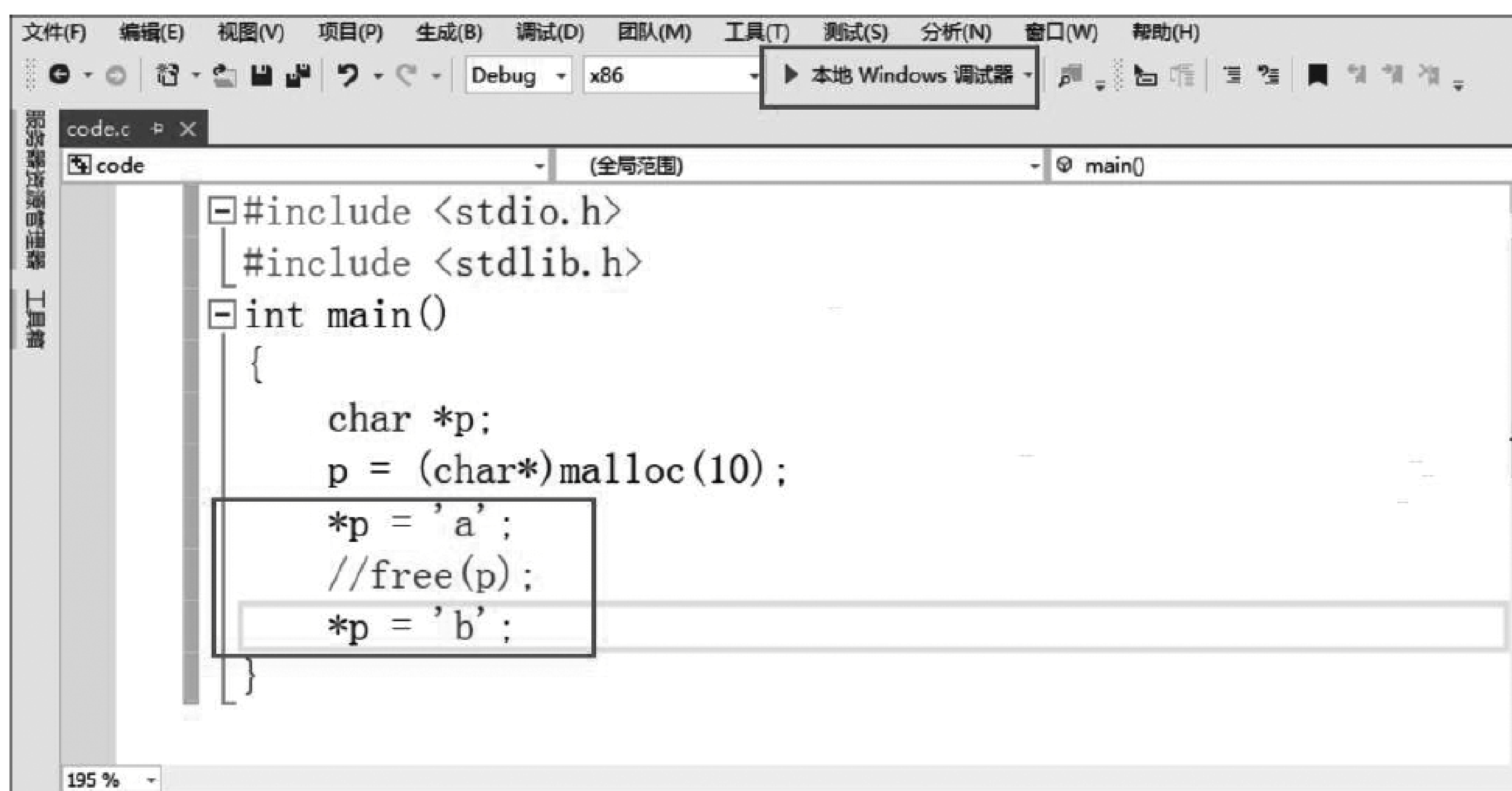


图 2-22　修改代码(2.1.1)

“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“释放后使用”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

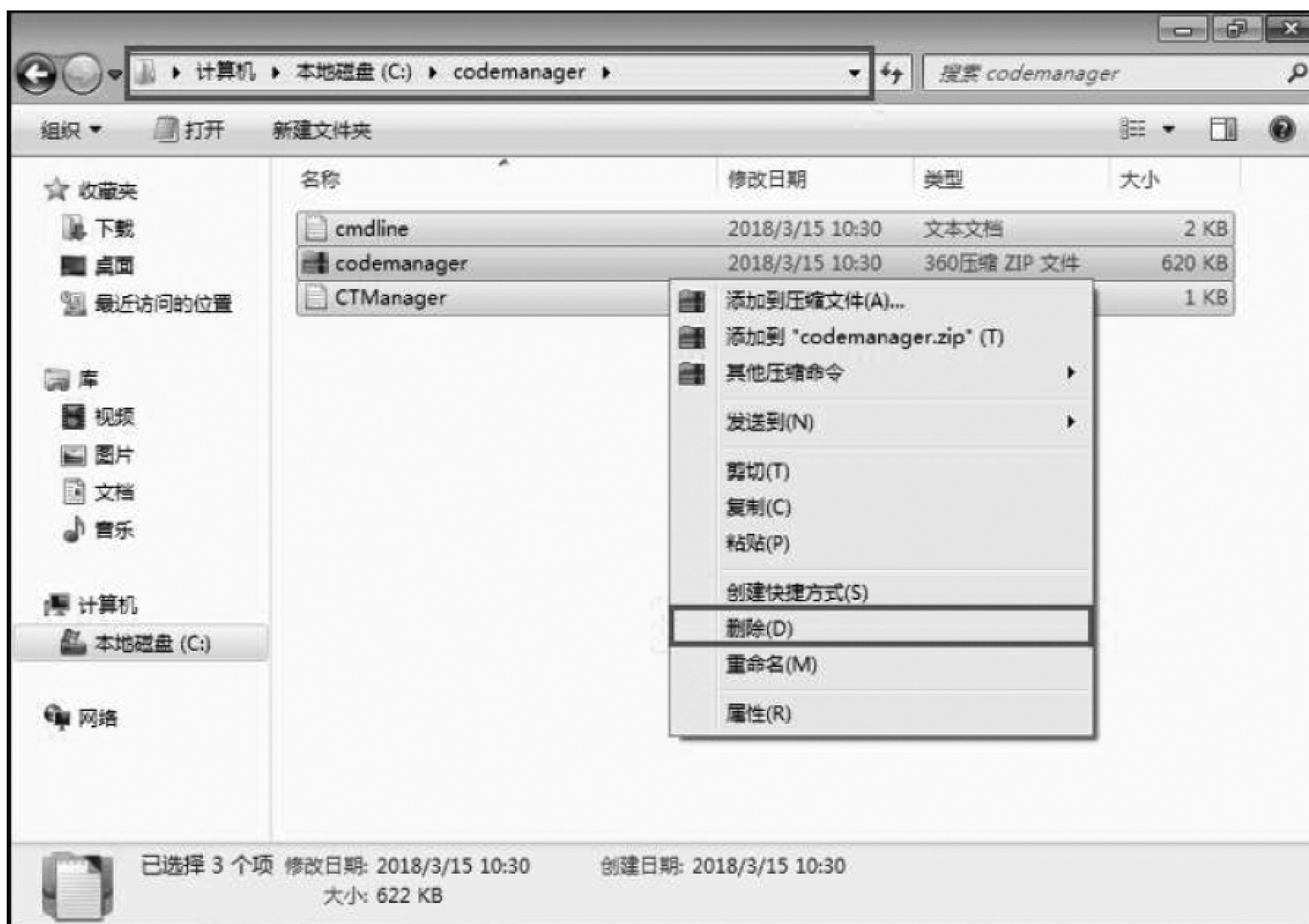


图 2-23　codemanager 文件夹

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 2-24 所示。



图 2-24　查看结果(2.1.1)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.2 代码返回栈地址缺陷检测实验

【实验目的】

确保所编写的代码中不存在返回栈地址操作。

【知识点】

“返回栈地址”缺陷、C/C++缺陷检测。

【场景描述】

A公司研发部工程师小王编写了一个字符串处理模块，将计算后的字符串进行返回，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：返回栈地址。

【实验原理】

C/C++中，函数内部的一切变量(函数内部局部变量、形参)都是在其被调用时才被分配内存单元。函数运行结束时，所有局部变量所占用的内存会被系统释放。形参和函数内部的局部变量的生命期和作用域都是在函数内部(static变量的生命期除外)。栈返回地址不可返回指向“栈内存”的“指针”，因为该内存在函数体结束时被自动销毁。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统1套。
- 主机终端：安装有VS 2015和Chrome浏览器的Windows 7 SP1主机1台。

【实验拓扑】

实验拓扑如图2-25所示。

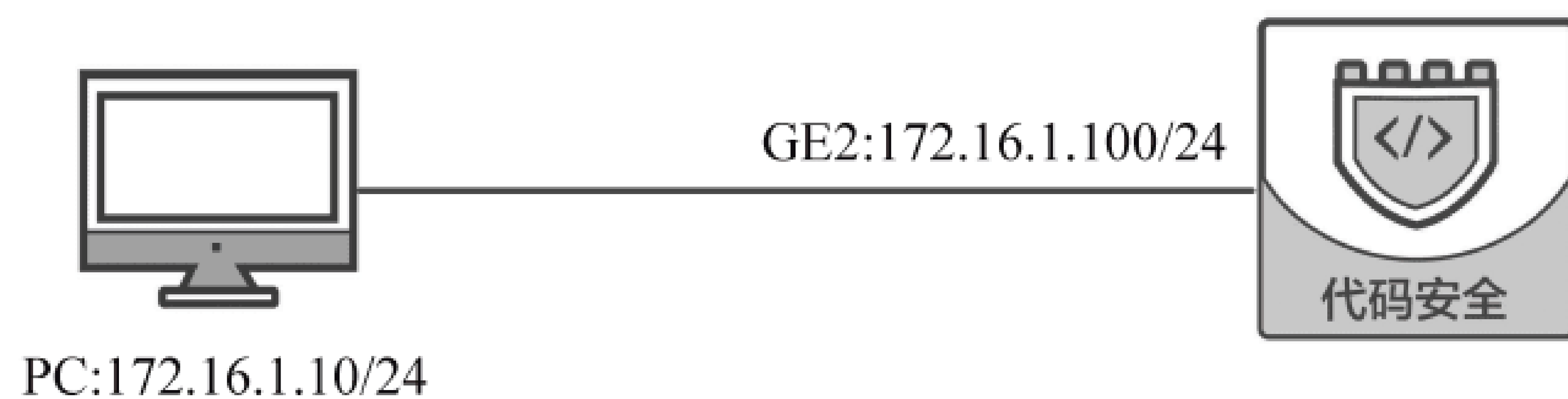


图2-25 代码返回栈地址缺陷检测实验拓扑图

【实验思路】

（1）建立项目文件。

（2）生成代码的中间表示文件。

（3）上传到代码卫士，检测出缺陷。

（4）依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）在终端机桌面双击 Visual Studio 2015，显示主界面。

（3）进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

（4）进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

（5）再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

（6）进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

（7）打开“C:\”下的 code.txt，复制其中的代码，如图 2-26 所示。

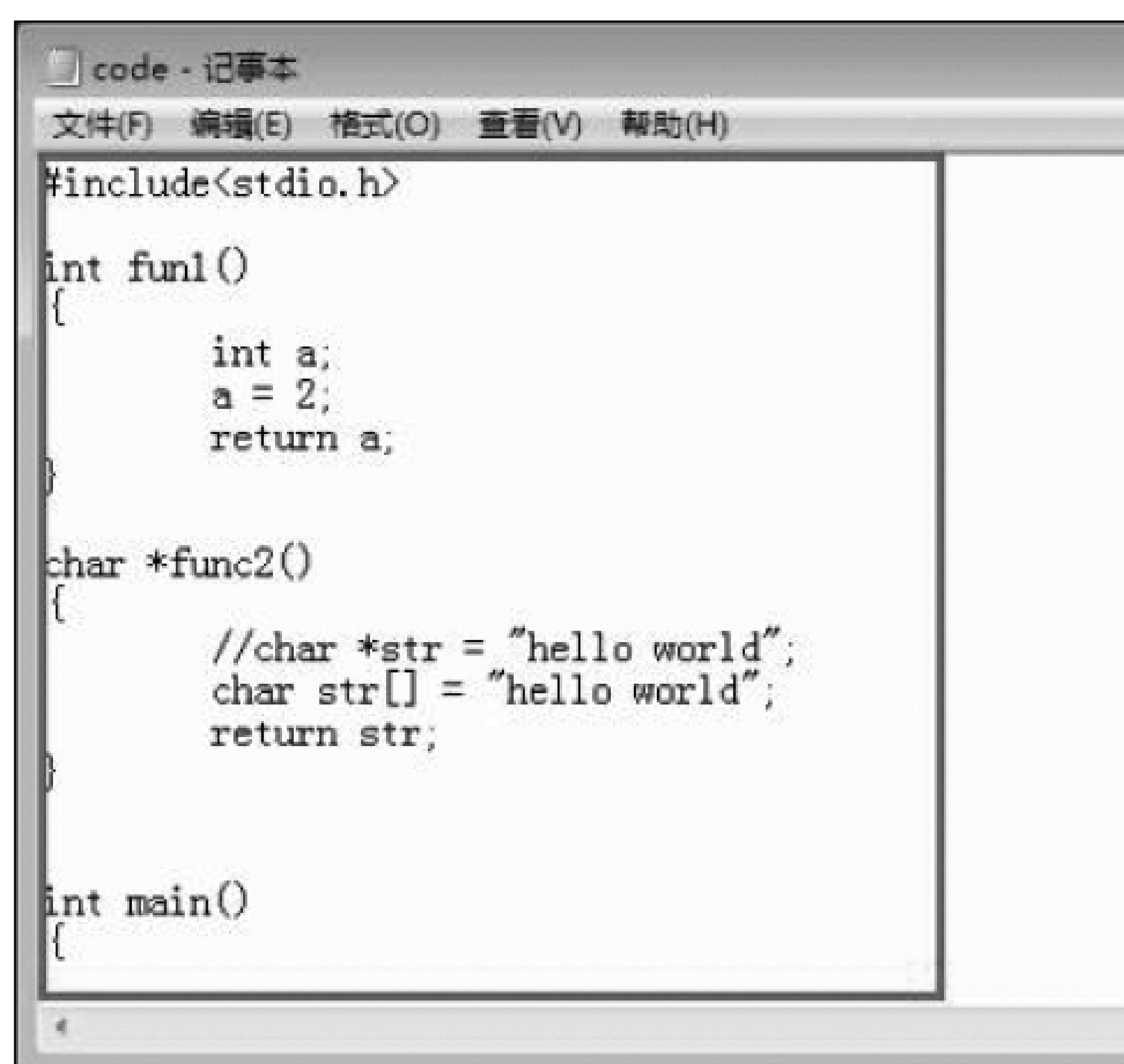


图 2-26　复制代码(2.1.2)

（8）进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

(9) 弹出“编译确认”对话框,单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(11) 进入“管理员:VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”输入“返回栈地址”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(20) 选择文件 codemanager,单击“打开”按钮。

(21) 返回“快速检测”界面,等待文件上传成功,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述与修复建议。

【实验结果】

1.检测出代码中释放后使用的缺陷

(1) 在学生本地机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 单击“+发起快速检测”按钮,查看列表所示检测结果。

(3) 找到“返回栈地址”,单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷,如图 2-27 所示。

(5) 打开“C:\code”文件夹,选择 code.sln,打开项目。

(6) 按图 2-28 中的方框部分修改代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015。

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件。



图 2-27　“快速检测”界面(2.1.2)

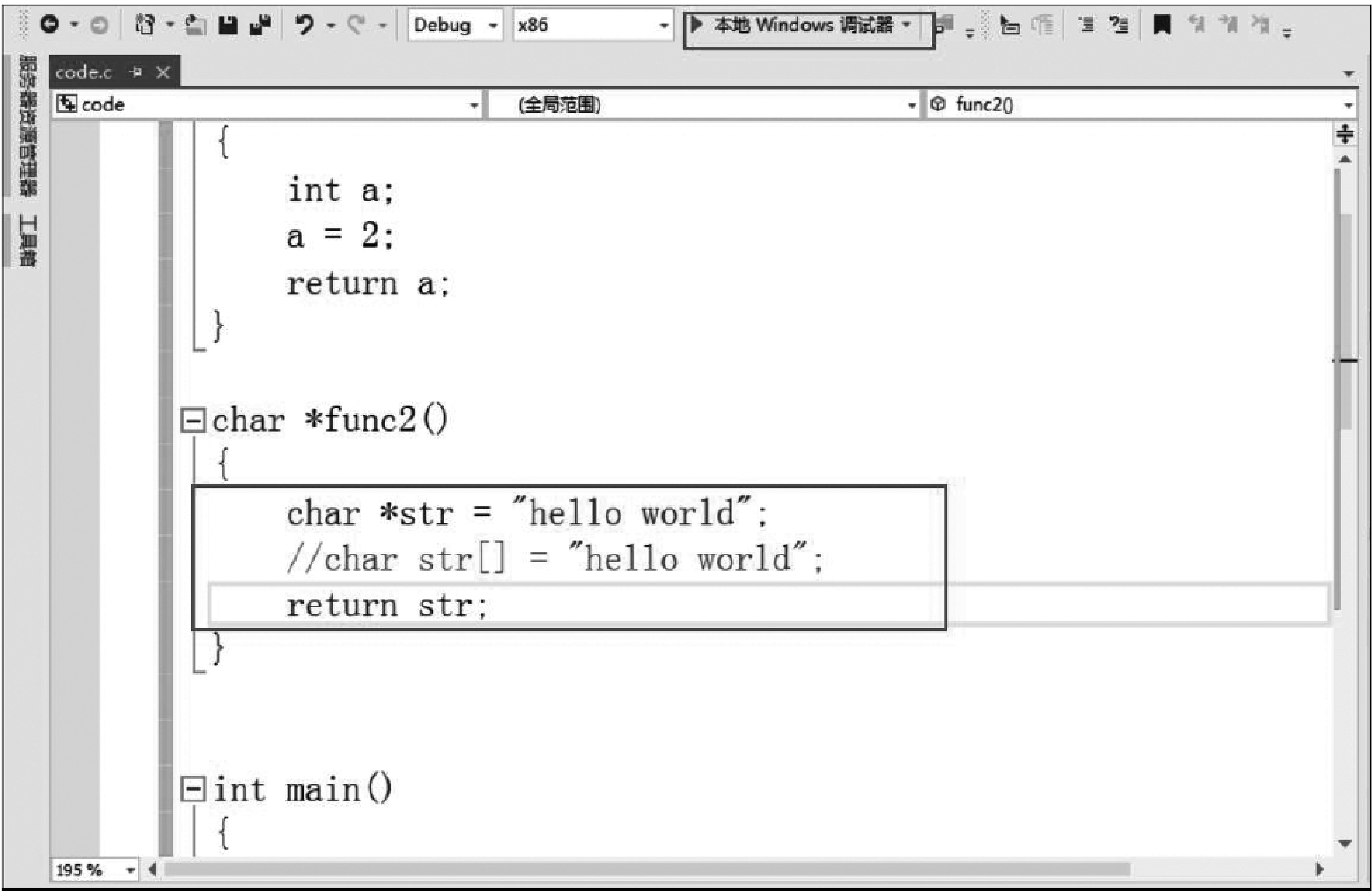


图 2-28　修改代码(2.1.2)

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(9) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”

命令。

(12) “任务名称”输入“返回栈地址”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 给出该缺陷的详细描述与修复建议

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-29 所示。



图 2-29　查看结果(2.1.2)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.3　空指针解引用缺陷检测实验

【实验目的】

确保所编写的代码中不存在空指针解引用。

【知识点】

“空指针引用”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个指针运算模块,对指针及指针指向的内存进行

处理，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：空指针解引用。

【实验原理】

一般导致程序崩溃的最重要原因之一就是试图解引用空指针。只要对 NULL 指针解引用，程序就会崩溃。

在 C 语言中对 void * 指针进行解引用会造成编译错误。

数值为 0 的整型常量表达式，或被转换为“void * ”类型的该种表达式，被称为空指针常量。如果一个空指针常量被转化为指针类型，该指针被称为空指针。

所以，C 语言空指针的本质是(void *)0。((void *)0) 本质上是将数字 0 强制转换为(void *)。0 其实就是地址，(void *) 只是说我们认为 0 这个地址中存储的类型是 void * ，也就是当前不知道、还未指定的。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-30 所示。

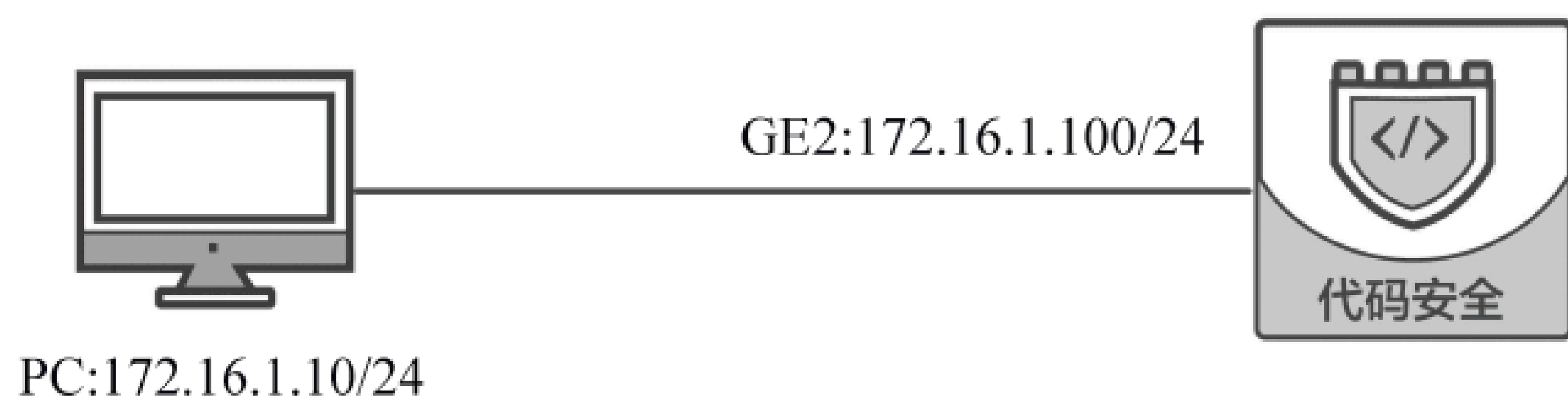


图 2-30　空指针解引用缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015,显示主界面。

(3) 进入主界面,选择"文件"→"新建"→"项目"命令,新建一个项目。

(4) 进入"新建项目"界面,在左侧"模板"选择"Visual C++"命令,界面中间选项选择"空项目",下侧"名称"选项框内输入 code,单击"位置"右侧的"浏览"按钮,选择"C:\",其他默认选择。单击"确定"按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上"解决方案资源管理器"小窗口,已经新建好 code 项目。右击"源文件",在弹出的快捷菜单中选择"添加"→"新建项"命令,添加一个新项。

(6) 进入"添加新项"界面,左侧"已安装"默认选择"Visual C++"命令,在界面中间选项选择"C++文件(.cpp)"。在下侧"名称"选项框内输入 code,单击"添加"按钮。

(7) 打开"C:\"下的 code.txt,复制其中的代码,如图 2-31 所示。

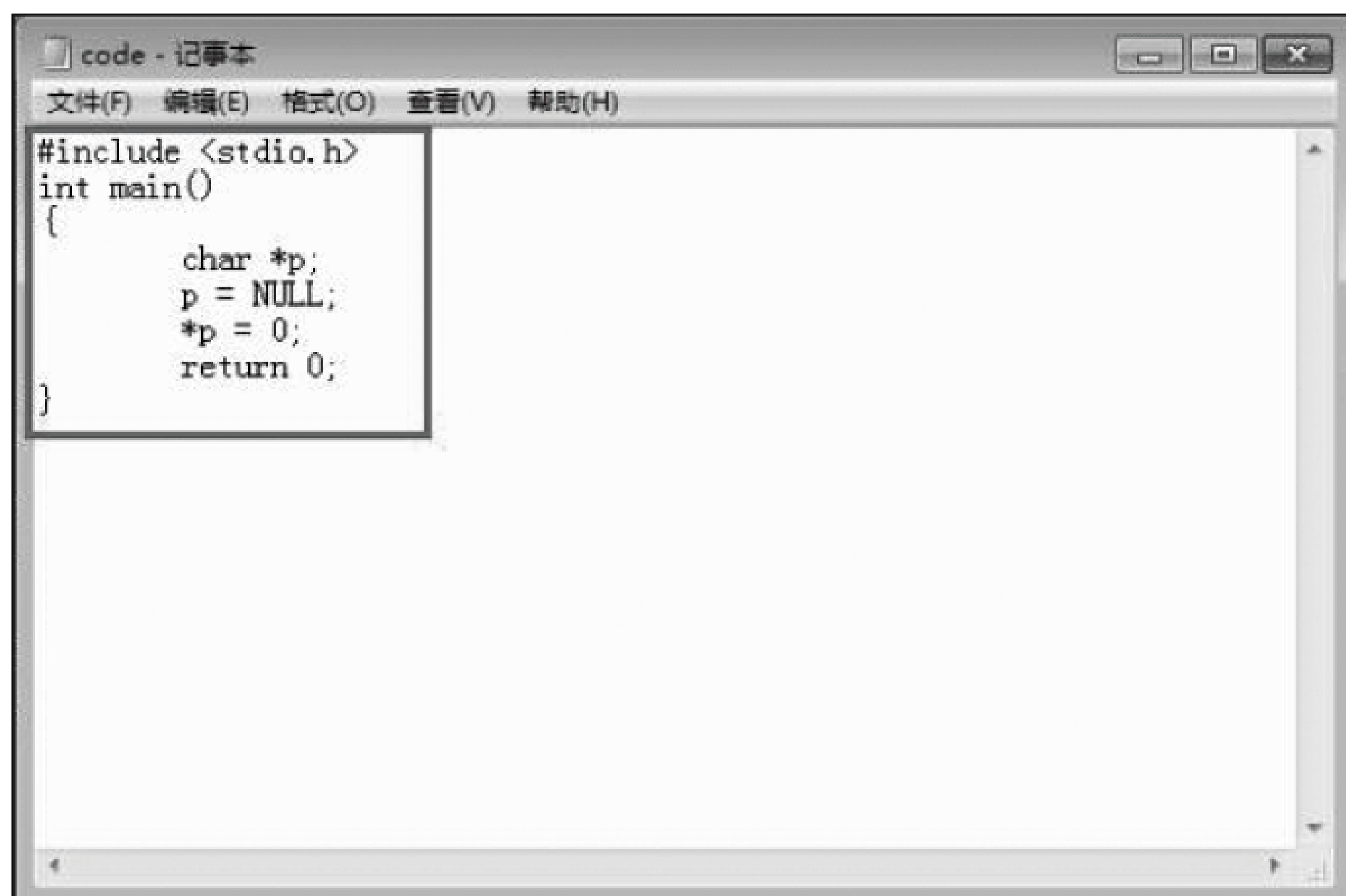


图 2-31 复制代码(2.1.3)

(8) 进入"Visual Studio 2015"代码编辑界面,在编辑框内粘贴代码,单击"本地 Windows 调试器"按钮。

(9) 弹出"编译确认"对话框,单击"是"按钮。

(10) 选择"开始"→"VS 2015 开发人员命令提示"命令,打开命令行界面。

(11) 进入"管理员: VS 2015 开发人员命令提示"界面,在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(12) 进入终端机"C:\"下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器,网址输入"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中,单击"高级"按钮。

(14) 在"高级"界面中单击"继续前往 172.16.1.100(不安全)"菜单命令。

（15）界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入 admin123！@＃，单击“登录”按钮。

（16）选择“快速检测”→“＋发起快速检测”命令。

（17）“任务名称”输入“空指针解引用”，“开发语言”选中“C/C++”单选框，勾选“缺陷检测模板[C/C++]”复选框。

（18）单击“浏览”按钮。

（19）单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

（20）选择文件 codemanager，单击“打开”按钮。

（21）返回“快速检测”界面，等待文件上传成功，单击“发起检测”按钮。

【实验预期】

（1）检测出代码中释放后使用的缺陷。

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

（1）在学生本地机打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123！@＃”，单击“登录”按钮。

（2）单击“＋发起快速检测”按钮，查看列表所示检测结果。

（3）找到“空指针解引用”，单击右侧的“缺陷审计”按钮。

（4）在左侧查看代码的缺陷，如图 2-32 所示。



图 2-32　“快速检测”界面(2.1.3)

(5) 打开“C:\code”文件夹,选择 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015,如图 2-33 所示。

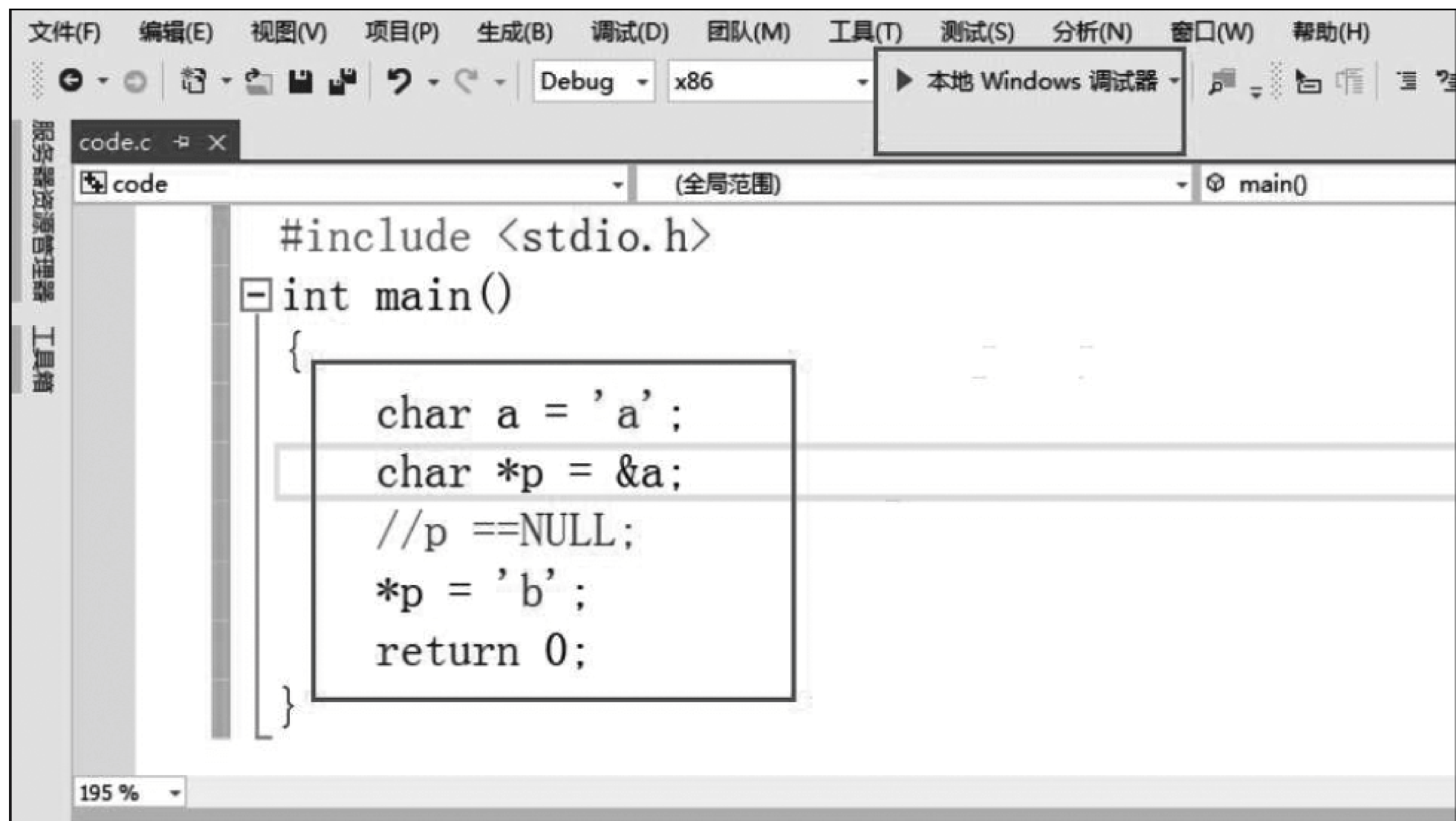


图 2-33　修改代码(2.1.3)

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\”下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面,选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“空指针解引用”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-34 所示。



图 2-34　查看结果(2.1.3)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.4　代码越界访问缺陷检测实验

【实验目的】

确保所编写的代码中不存在越界访问问题。

【知识点】

“越界访问”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个数组运算模块，对数组中的元素进行运算处理，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：越界访问。

【实验原理】

C 语言数组不会自动扩容，当下标小于零或大于等于数组长度时，就会发生越界(Out Of Bounds，OOB)，访问到数组以外的内存。如果下标小于零，会发生下限越界(Off Normal Lower，ONL)；如果下标大于等于数组长度，会发生上限越界(Off Normal Upper，ONU)。

C 语言为了提高效率，并不会对越界行为进行检查，即使越界了，也能够正常编译，只

有在运行期间才可能会发生问题。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-35 所示。

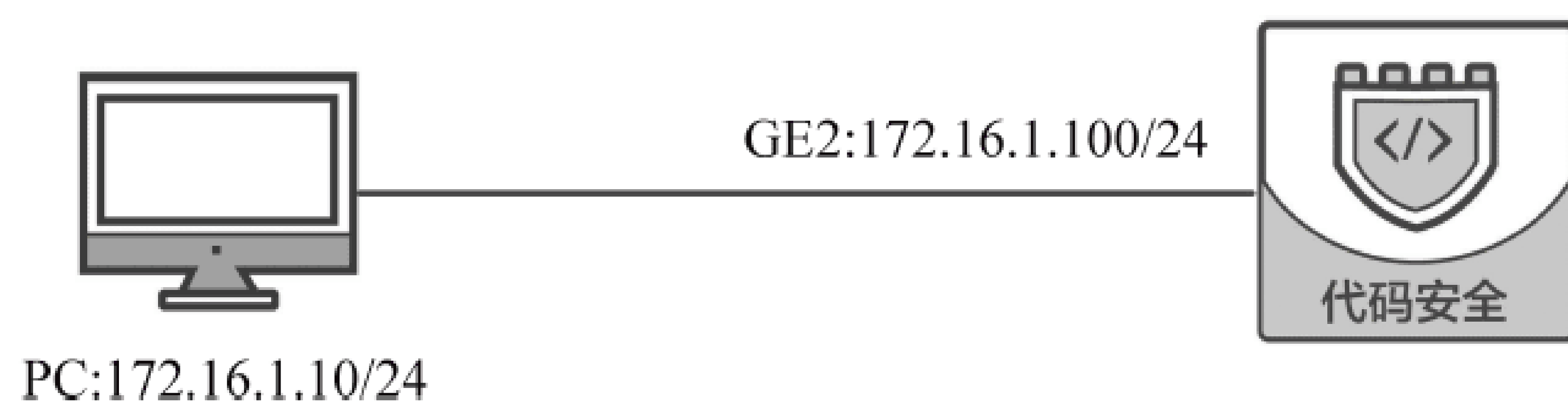


图 2-35　代码越界访问缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入“Visual Studio 2015”主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

(7) 打开“C:\”下的 code.txt，复制其中的代码，如图 2-36 所示。

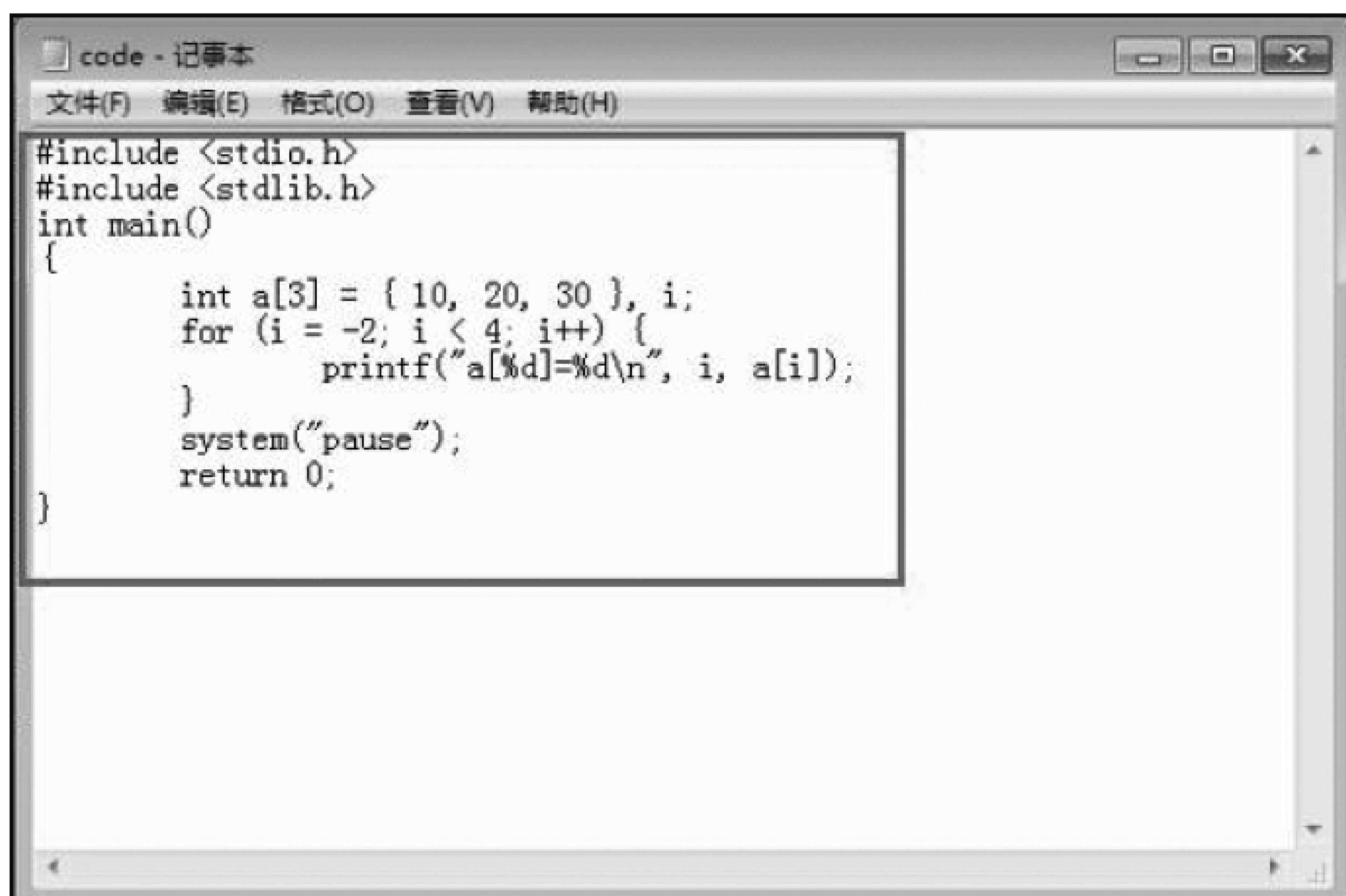


图 2-36　复制代码(2.1.4)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮，运行结束后关闭 VS 2015。

(9) 弹出“编译确认”对话框，单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(11) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”填写“越界访问”，“开发语言”选择“C/C++”，勾选“缺陷检测模板[C/C++]”。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(20) 选择文件 codemanager，单击“打开”按钮。

(21) 返回“快速检测”界面，等待文件上传成功，单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

（1）在学生本地机打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@＃”，单击“登录”按钮。

（2）单击“＋发起快速检测”按钮，查看列表所示检测结果。

（3）找到“越界访问”，单击右侧的“缺陷审计”按钮。

（4）在左侧查看代码的缺陷，如图 2-37 所示。



图 2-37 “快速检测”界面(2.1.4)

（5）打开“C:\code”文件夹，选择 code.sln，打开项目。

（6）在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，运行结束后关闭 VS 2015，如图 2-38 所示。

（7）打开“C:\codemanager”文件夹，删除其中的所有文件。

（8）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

（9）进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

（10）进入学生机“C:\”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

（11）进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”命令。

（12）“任务名称”输入“越界访问”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

（13）单击“浏览”按钮。

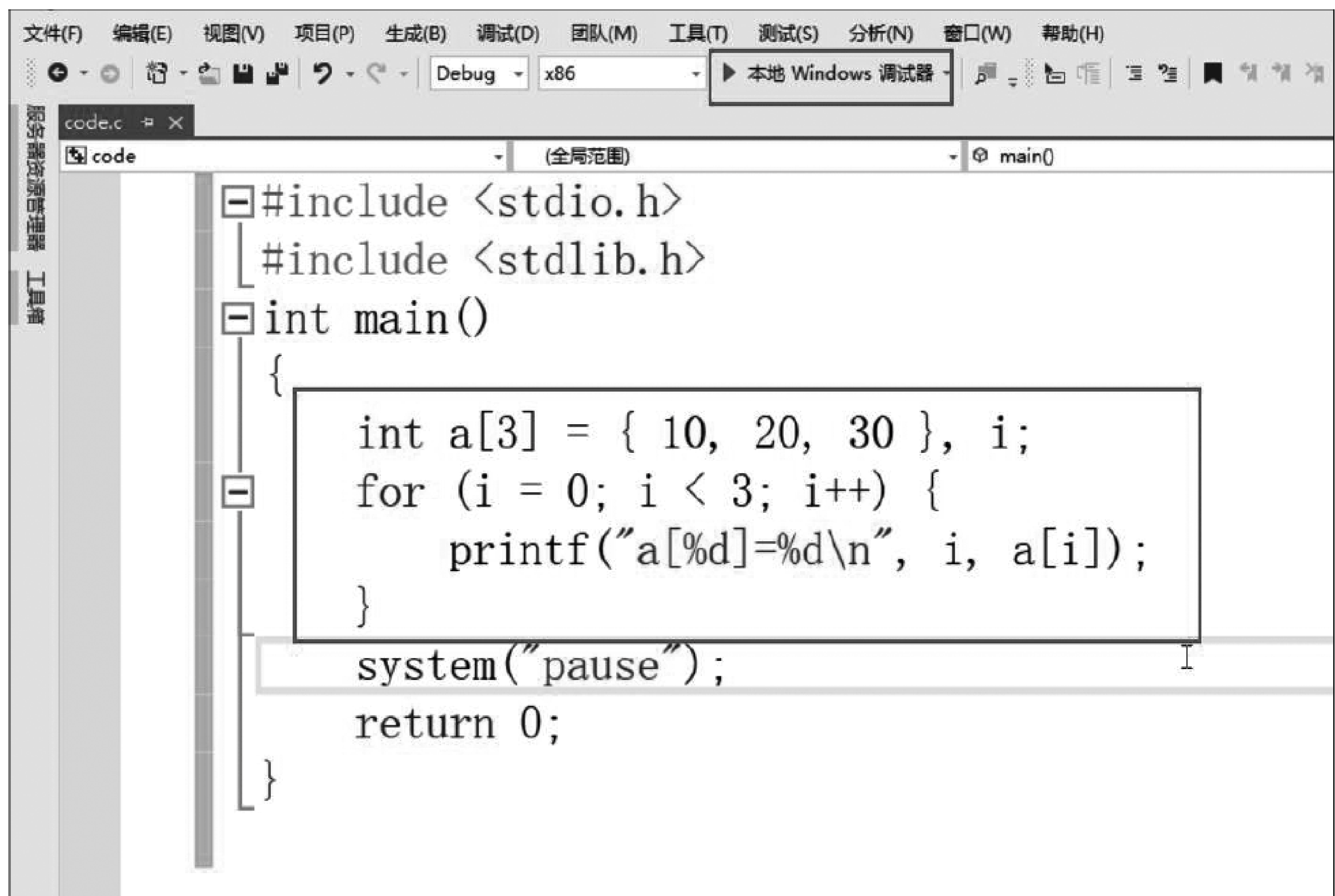


图 2-38　修改代码(2.1.4)

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2.给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后，选择任务右侧的“缺陷审计”命令。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 2-39 所示。



图 2-39　查看结果(2.1.4)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.5 无符号整数回绕缺陷检测实验

【实验目的】

确保所编写的代码中不存在无符号整数回绕操作。

【知识点】

“无符号整数回绕”缺陷、C/C++缺陷检测。

【场景描述】

A公司研发部工程师小王编写了一个整型运算模块，对传入的整型数据进行数学运算，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：无符号整数回绕。

【实验原理】

“C/C++”中，涉及无符号操作数的计算永远不会溢出，因为无法由最终的无符号整数类型表示的结果，将会根据这种最终类型可以表示的最大值加溢出数据执行求模操作。这个行为更通俗的说法是无符号整数将会回绕。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统1套。
- 主机终端：安装有VS 2015和Chrome浏览器的Windows 7 SP1主机1台。

【实验拓扑】

实验拓扑如图2-40所示。

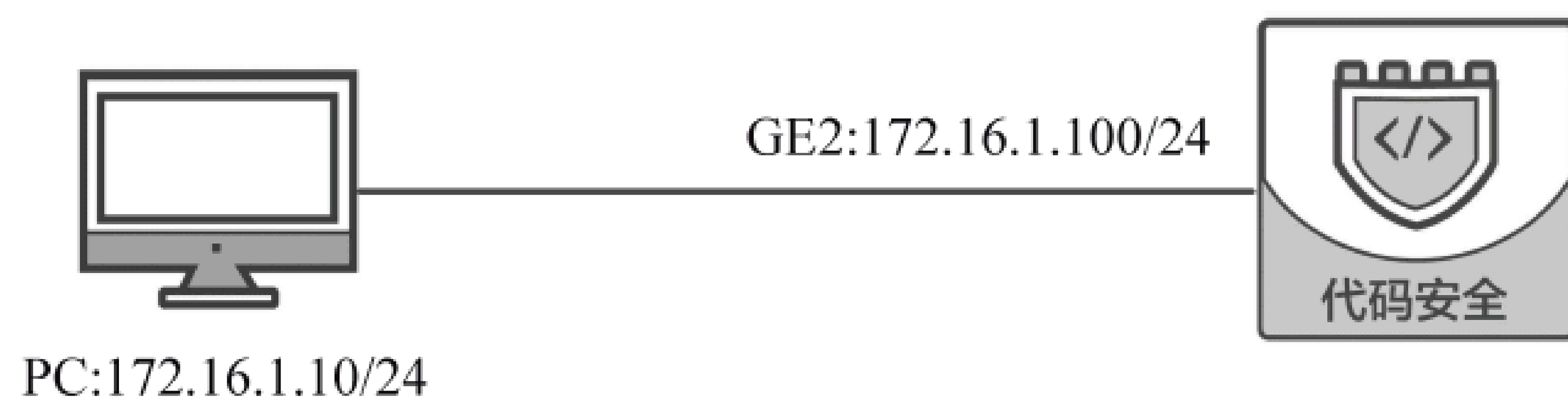


图2-40 无符号整数回绕缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015,显示主界面。

(3) 进入主界面,选择“文件”→“新建”→“项目”命令,新建一个项目。

(4) 进入“新建项目”界面,在左侧“模板”选择“Visual C++”命令,界面中间选项选择“空项目”,下侧“名称”选项框内输入 code,单击“位置”右侧的“浏览”按钮,选择“C:\”,其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上“解决方案资源管理器”小窗口,已经新建好 code 项目。右击“源文件”,在弹出的快捷菜单中选择“添加”→“新建项”命令,添加一个新项。

(6) 进入“添加新项”界面,左侧“已安装”默认选择“Visual C++”命令,在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code,单击“添加”按钮。

(7) 打开“C:\”的 code.txt,复制其中的代码,如图 2-41 所示。

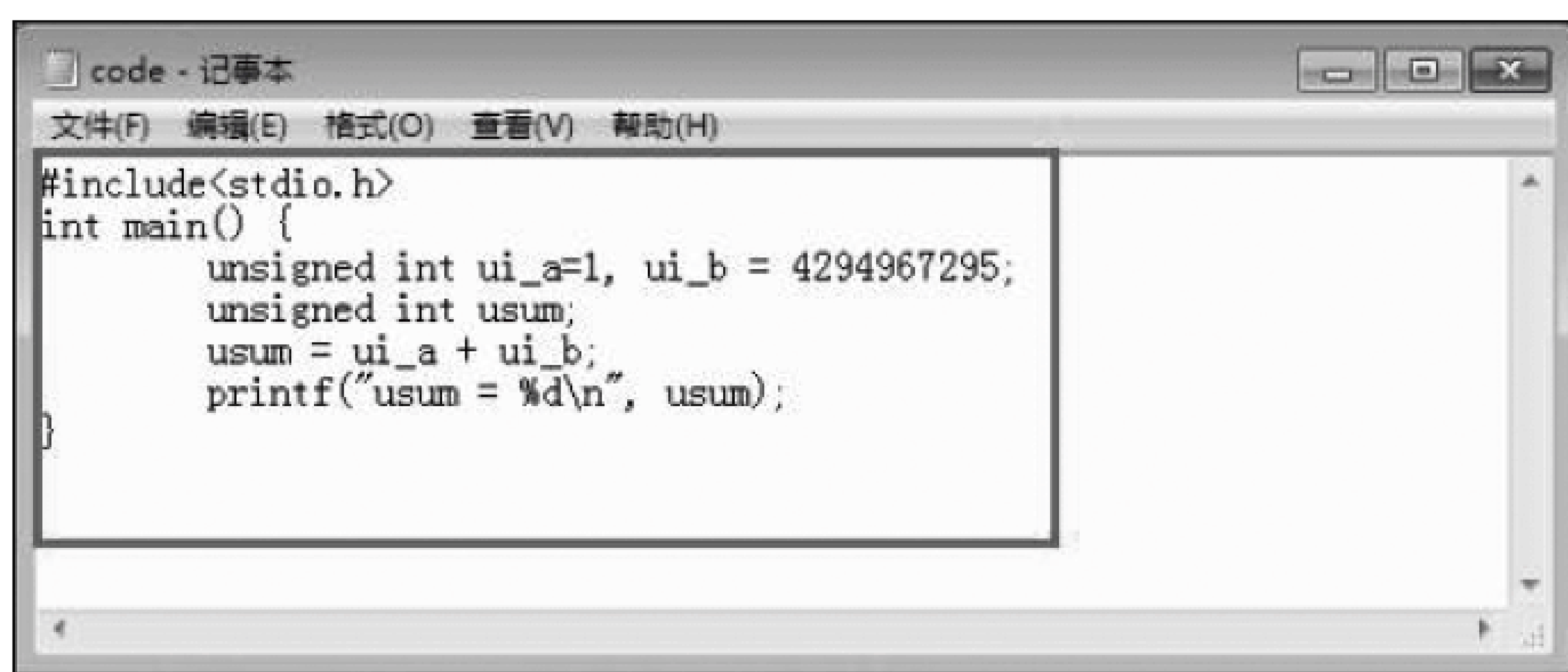


图 2-41　复制代码(2.1.5)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015。

(9) 弹出“编译确认”对话框,单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(11) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”菜单命令。

(17) “任务名称”输入“无符号整数回绕”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(20) 选择文件 codemanager,单击“打开”按钮。

(21) 返回“快速检测”界面,等待文件上传成功,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

(1) 在学生本地机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 单击“+发起快速检测”按钮,查看列表所示检测结果。

(3) 找到“无符号整数回绕”,单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷,如图 2-42 所示。

(5) 打开“C:\code”文件夹,选择 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015,如图 2-43 所示。

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\”下的 codemanager,显示中间文件 codemanager.zip 生成



图 2-42　“快速检测”界面(2.1.5)

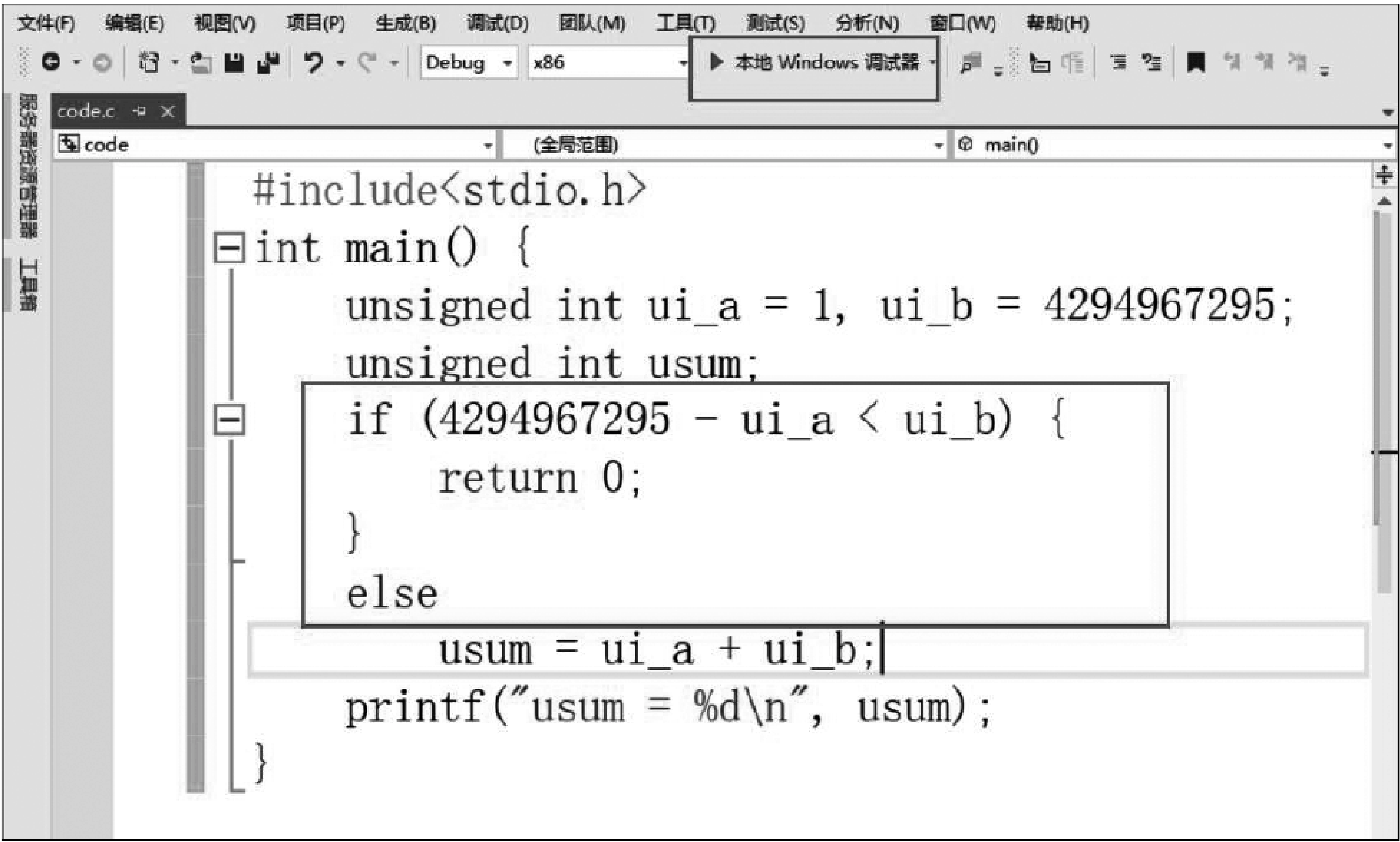


图 2-43　修改代码(2.1.5)

成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”命令。

(12)“任务名称”输入“无符号整数回绕”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-44 所示。



图 2-44　查看结果(2.1.5)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.6　字符串缺少终止符缺陷检测实验

【实验目的】

确保所编写的代码中不存在字符串缺少终止符的情况。

【知识点】

“字符串缺少终止符”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个字符串运算模块,调用库函数对字符串进行处理,需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:字符串缺少终止符。

【实验原理】

C 语言中,以字符数组作为字符串,正确的以'\0'字符作为结尾的情况是指在数组最

后一个元素处或在它之前存在一个空终结符。如果一个字符串没有以空字符结尾，程序无法准确判断字符串终结的位置，导致在数组边界之外读取或写入数据。例如：

```
char a[16] = {"0123456789abcde"};
a[15] = 'f';
printf("%s",a);
```

在 printf 函数输出时，输出完 0123456789abcdef 后，还未遇到终止符，所以函数还会有未知的输出。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-45 所示。

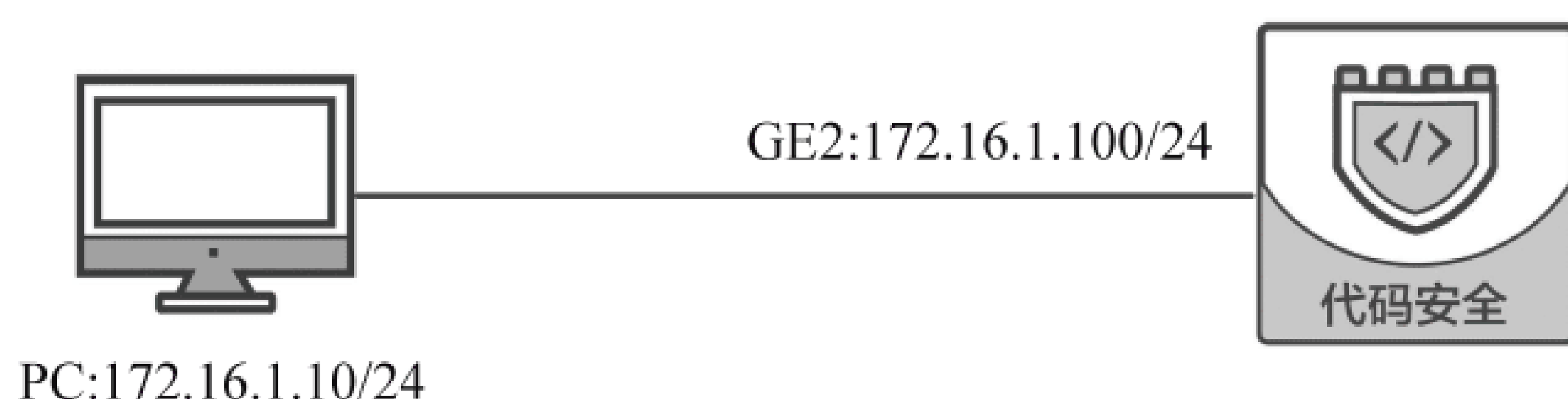


图 2-45　字符串缺少终止符缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择"文件"→"新建"→"项目"命令，新建一个项目。

(4) 进入"新建项目"界面，在左侧"模板"选择"Visual C++"命令，界面中间选项选择"空项目"，下侧"名称"选项框内输入 code，单击"位置"右侧的"浏览"按钮，选择"C:\"，其他默认选择。单击"确定"按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上"解决方案资源管理器"小窗口，已经新建好 code 项目。右击"源文件"，在弹出的快捷菜单中选择"添加"→"新建项"

命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

(7) 打开“C:\”的 code.txt，复制其中的代码，如图 2-46 所示。



图 2-46 复制代码(2.1.6)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

(9) 弹出编译确认对话框，单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”菜单命令，打开命令行界面。

(11) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”菜单命令。

(17) “任务名称”输入“字符串缺少终止符”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(20) 选择文件 codemanager，单击“打开”按钮。

(21) 返回“快速检测”界面，等待上传成功，单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中字符串缺少终止符的缺陷。

(2) 给出该缺陷的详细描述与修复建议。

【实验结果】

1. 检测出代码中字符串缺少终止符的缺陷

(1) 在终端机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 单击“+发起快速检测”按钮,查看列表所示检测结果。

(3) 找到“字符串缺少终止符”,单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷,如图 2-47 所示。



图 2-47 “快速检测”界面(2.1.6)

(5) 打开“C:\code”文件夹,选择 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,如图 2-48 所示。

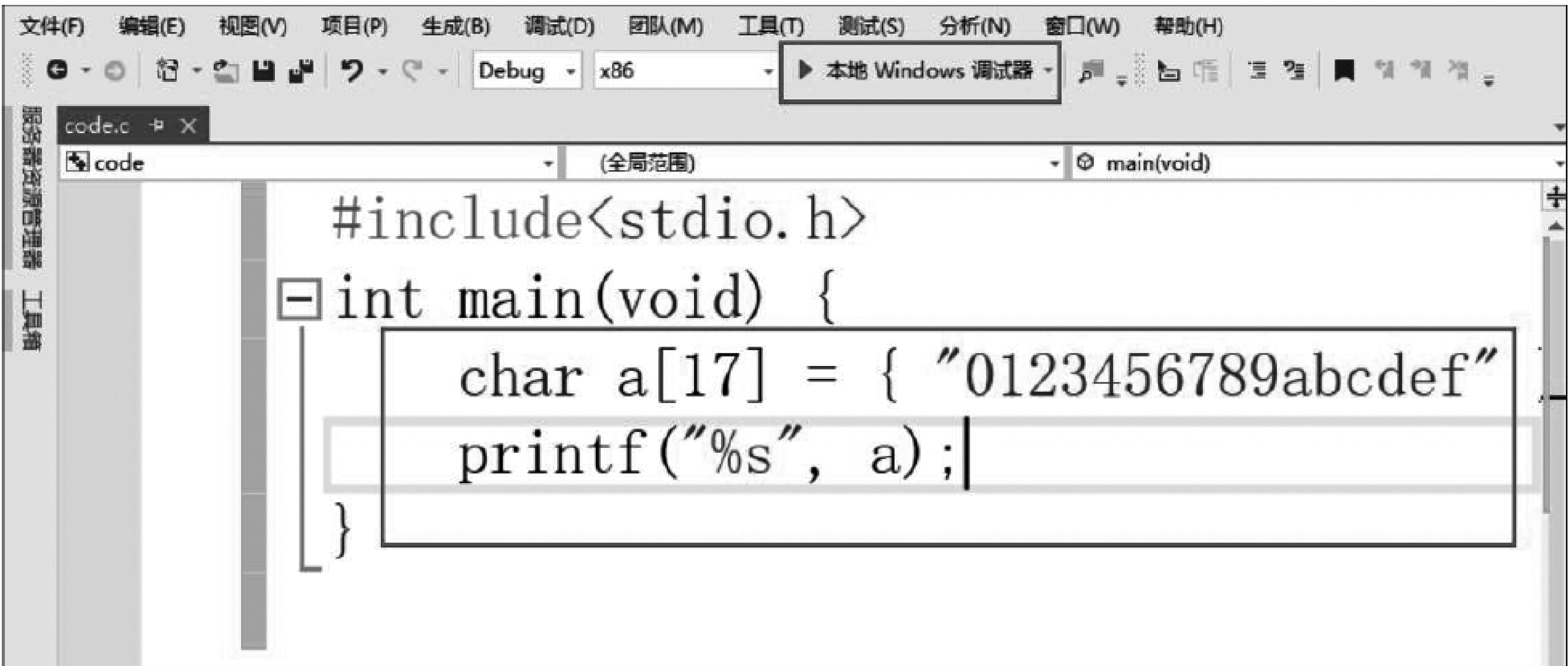


图 2-48 修改代码(2.1.6)

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”菜单命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入终端机“C:\ ”下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面,选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“字符串缺少终止符”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,等待上传成功,单击“发起检测”按钮。

2. 给出该缺陷的详细描述与修复建议

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-49 所示。



图 2-49 查看结果(2.1.6)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考字符串缺少终止符会引起的后果。

2.1.7　代码在 scanf 函数中没有对%s 格式符进行宽度限制缺陷检测实验

【实验目的】

确保所编写的代码中不存在 scanf 函数中没有对%s 格式符进行宽度限制的情况。

【知识点】

“在 scanf 函数中没有对%s 格式符进行宽度限制”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个 IO 模块，调用 scanf 对用户输入的数据进行处理，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：在 scanf 函数中没有对%s 格式符进行宽度限制。

【实验原理】

在 C 语言中，scanf 函数是一个格式输入函数，即按用户指定的格式从标准输入流 stdin 中把数据输入指定的变量之中。如果在 scanf 函数中没有对%s 格式符进行宽度限制，则可能造成输入超出字符串参数的长度限制，导致字符串溢出。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-50 所示。

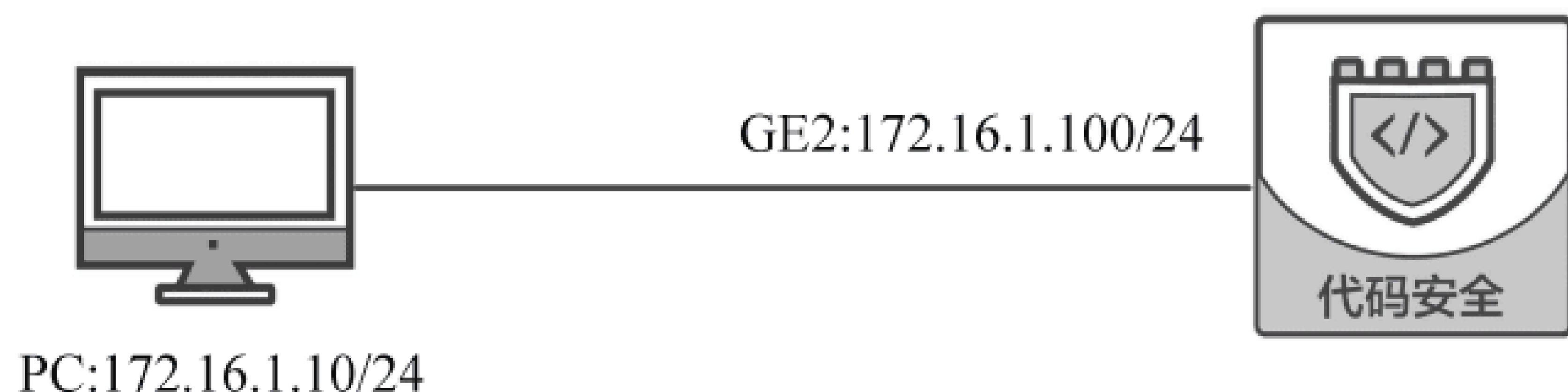


图 2-50　代码在 scanf 函数中没有对%s 格式符进行宽度限制缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015 按钮,显示主界面。

(3) 进入主界面,选择"文件"→"新建"→"项目"命令,新建一个项目。

(4) 进入"新建项目"界面,在左侧"模板"选择"Visual C++"命令,界面中间选项选择"空项目",下侧"名称"选项框内输入 code,单击"位置"右侧的"浏览"按钮,选择"C:\",其他默认选择。单击"确定"按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上"解决方案资源管理器"小窗口,已经新建好 code 项目。右击"源文件",在弹出的快捷菜单中选择"添加"→"新建项"命令,添加一个新项。

(6) 进入"添加新项"界面,左侧"已安装"默认选择"Visual C++"命令,在界面中间选项选择"C++文件(.cpp)"。在下侧"名称"选项框内输入 code,单击"添加"按钮。

(7) 打开"C:\ "下的 code.txt,复制其中的代码,如图 2-51 所示。

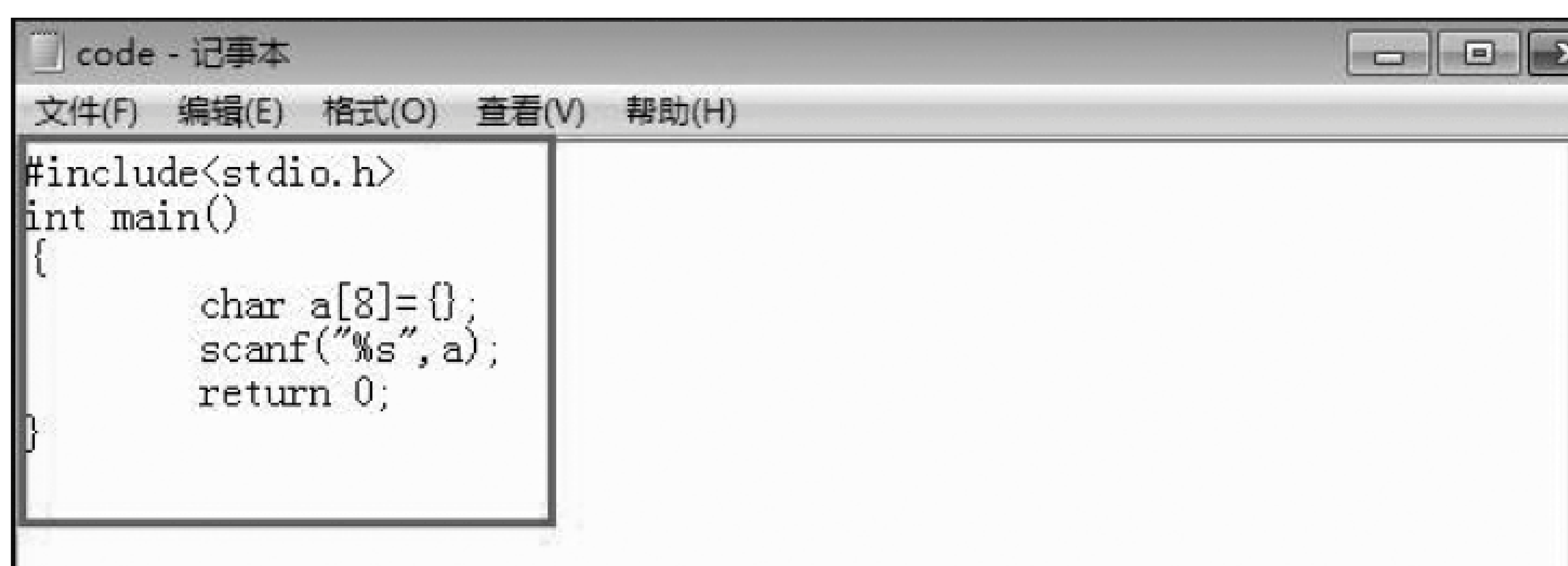


图 2-51　复制代码(2.1.7)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,右击"解决方案资源管理器"窗口中的 code 项目,在弹出的快捷菜单中选择"属性"命令。

(9) 选择"配置属性"中的"C/C++"下的"预处理器"命令,如图 2-52 所示。



图 2-52　预处理器

(10) 选择“预处理器定义”命令，如图 2-53 所示。

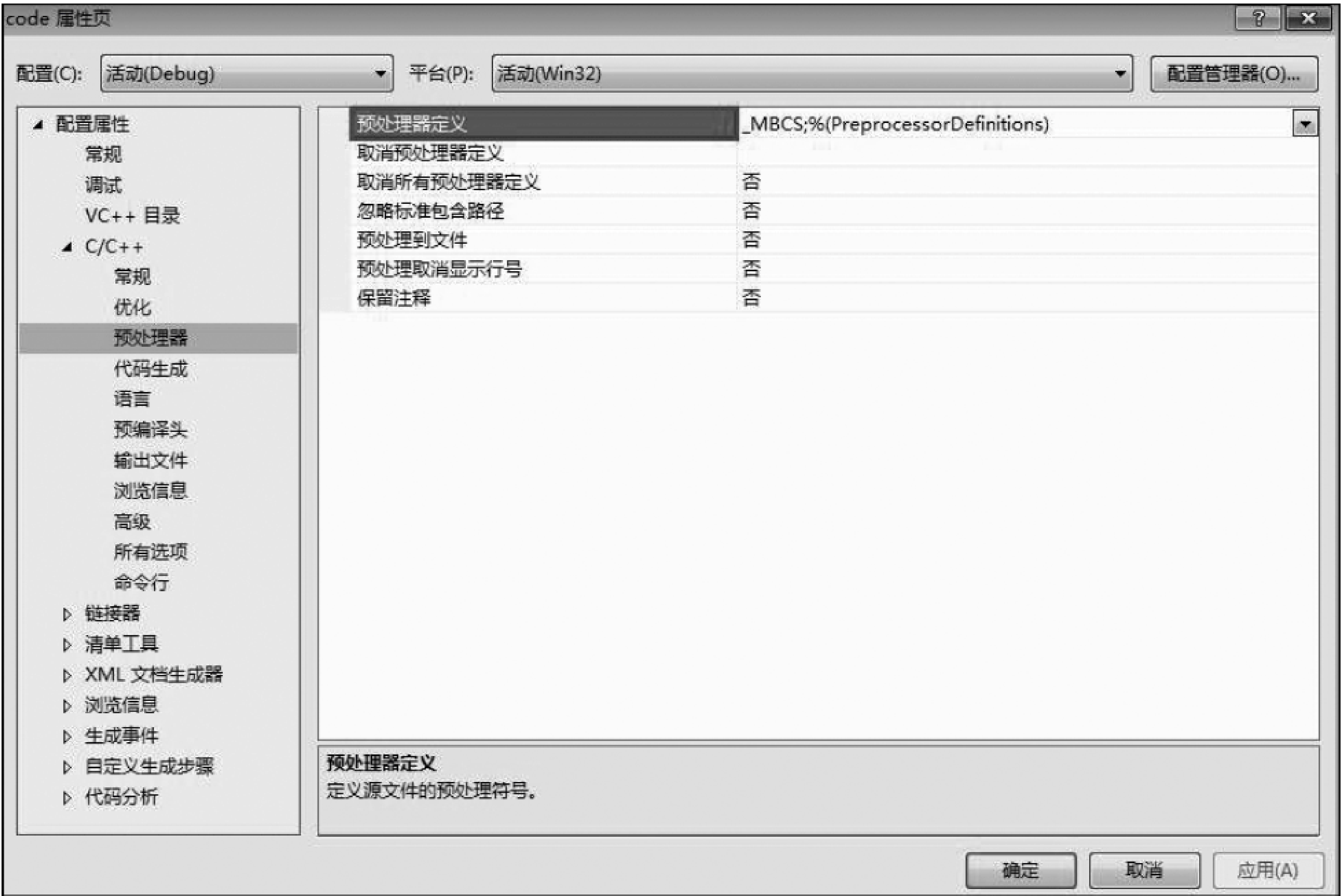


图 2-53　预处理器定义

(11) 单击右方的“下拉”按钮，如图 2-54 所示。

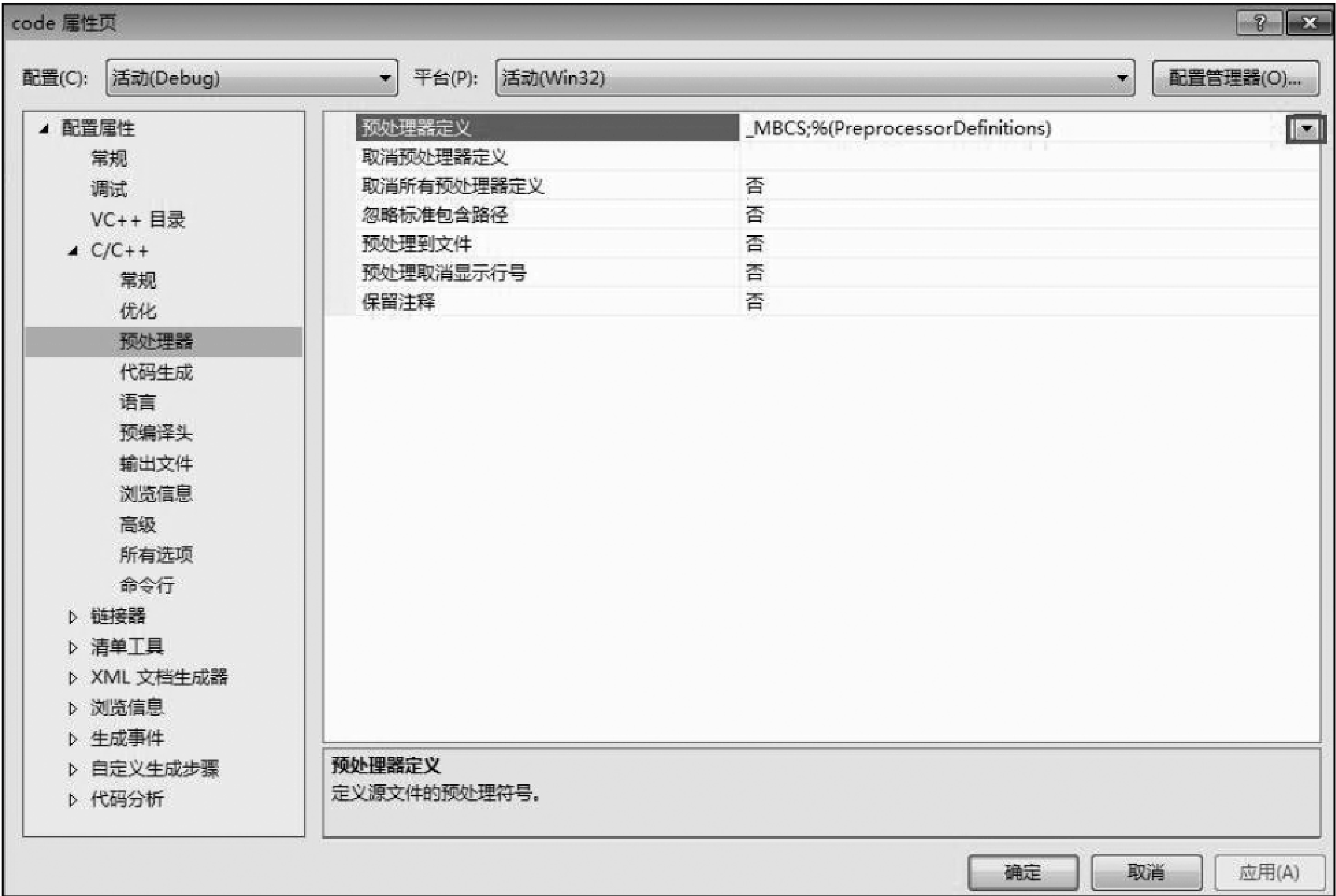


图 2-54　单击“下拉”按钮

(12) 选择“编辑”命令，如图 2-55 所示。

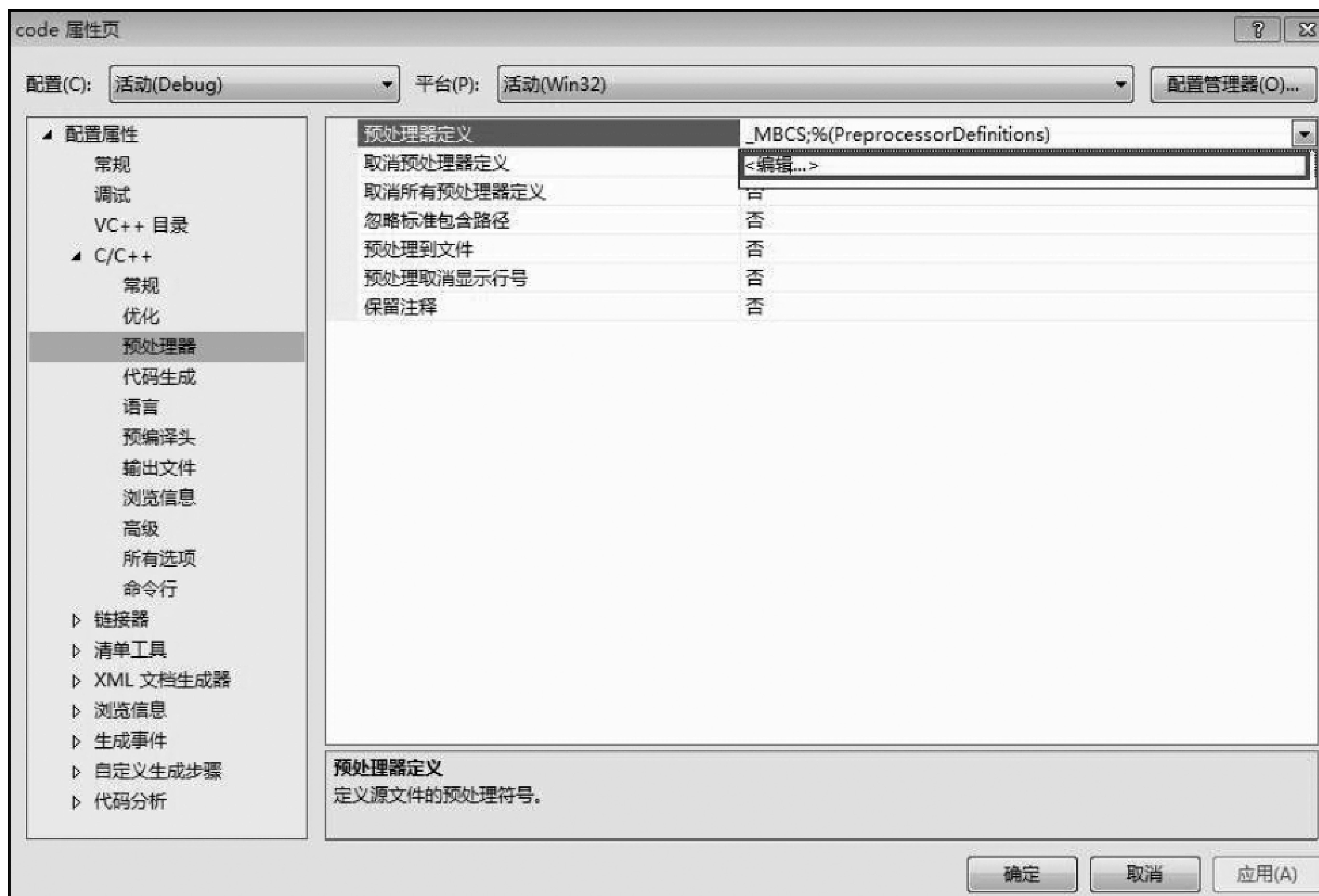


图 2-55　单击“编辑”按钮

(13) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮，如图 2-56 所示。

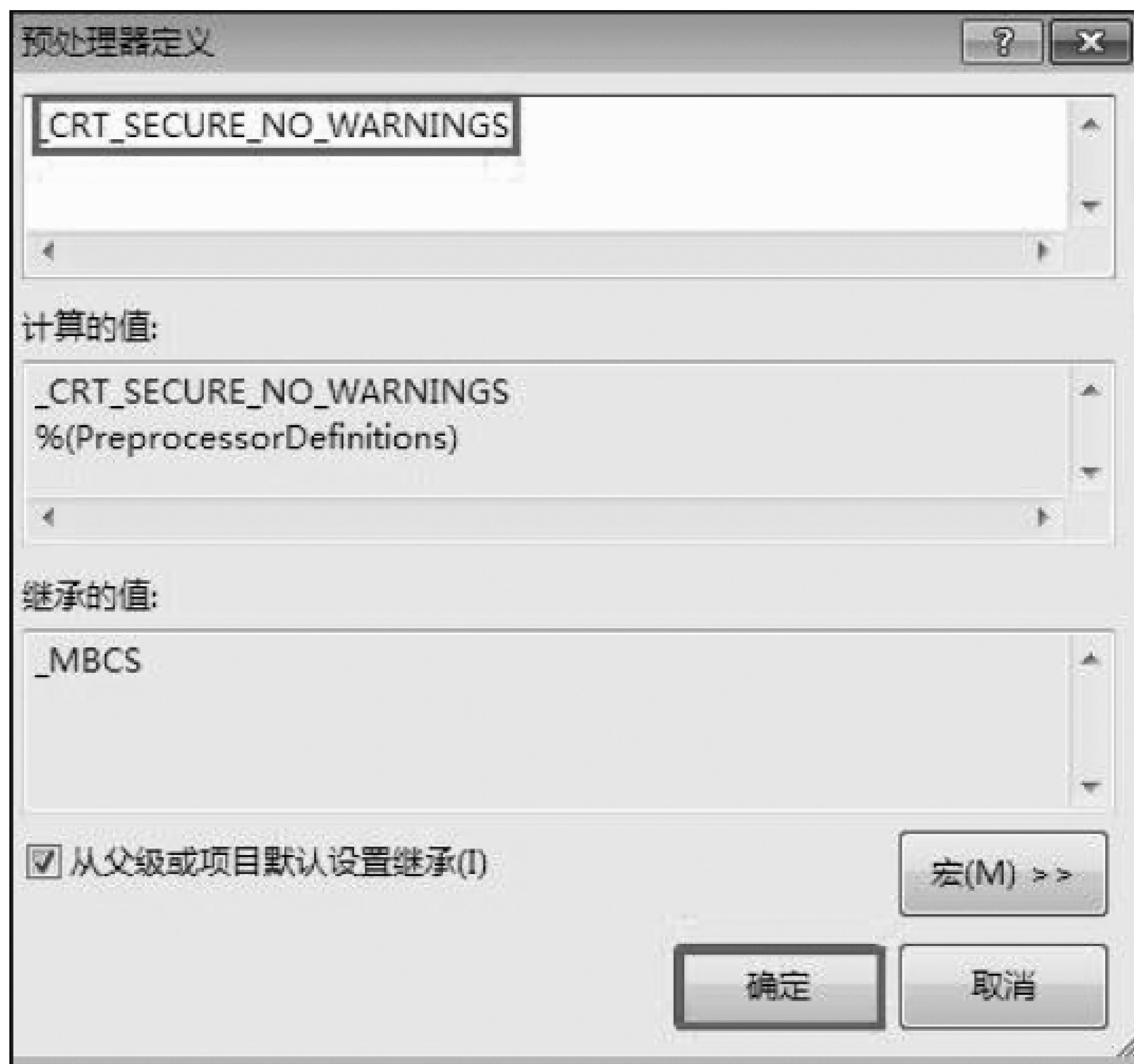


图 2-56　预处理器定义界面

(14) 单击“code 属性页”右上方的“关闭”按钮，关闭“code 属性页”，如图 2-57 所示。

(15) 单击“本地 Windows 调试器”按钮。

(16) 弹出“编译确认”对话框，单击“是”按钮。

(17) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。



图 2-57　关闭“code 属性页”

(18) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(19) 进入终端机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(20) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(21) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(22) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(23) 选择“快速检测”→“+发起快速检测”命令。

(24) “任务名称”输入“scanf 没有宽度限制”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(25) 单击“浏览”按钮。

(26) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(27) 选择文件 codemanager，单击“打开”按钮。

(28) 返回“快速检测”界面，单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中在 scanf 函数中没有对“%s”格式进行宽度限制的缺陷。

(2) 对比修改前后检测结果。

【实验结果】

1. 检测出代码中在 scanf 函数中没有对“%s”格式进行宽度限制的缺陷

(1) 在终端机打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@#”，单击“登录”按钮。

(2) 单击“快速检测”按钮，查看列表所示检测结果。

(3) 找到“scanf 没有宽度限制”，单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷，如图 2-58 所示。

(5) 打开“C:\code”文件夹，双击 code.sln，打开项目。



图 2-58　“快速检测”界面(2.1.7)

(6) 在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 2-59 所示。

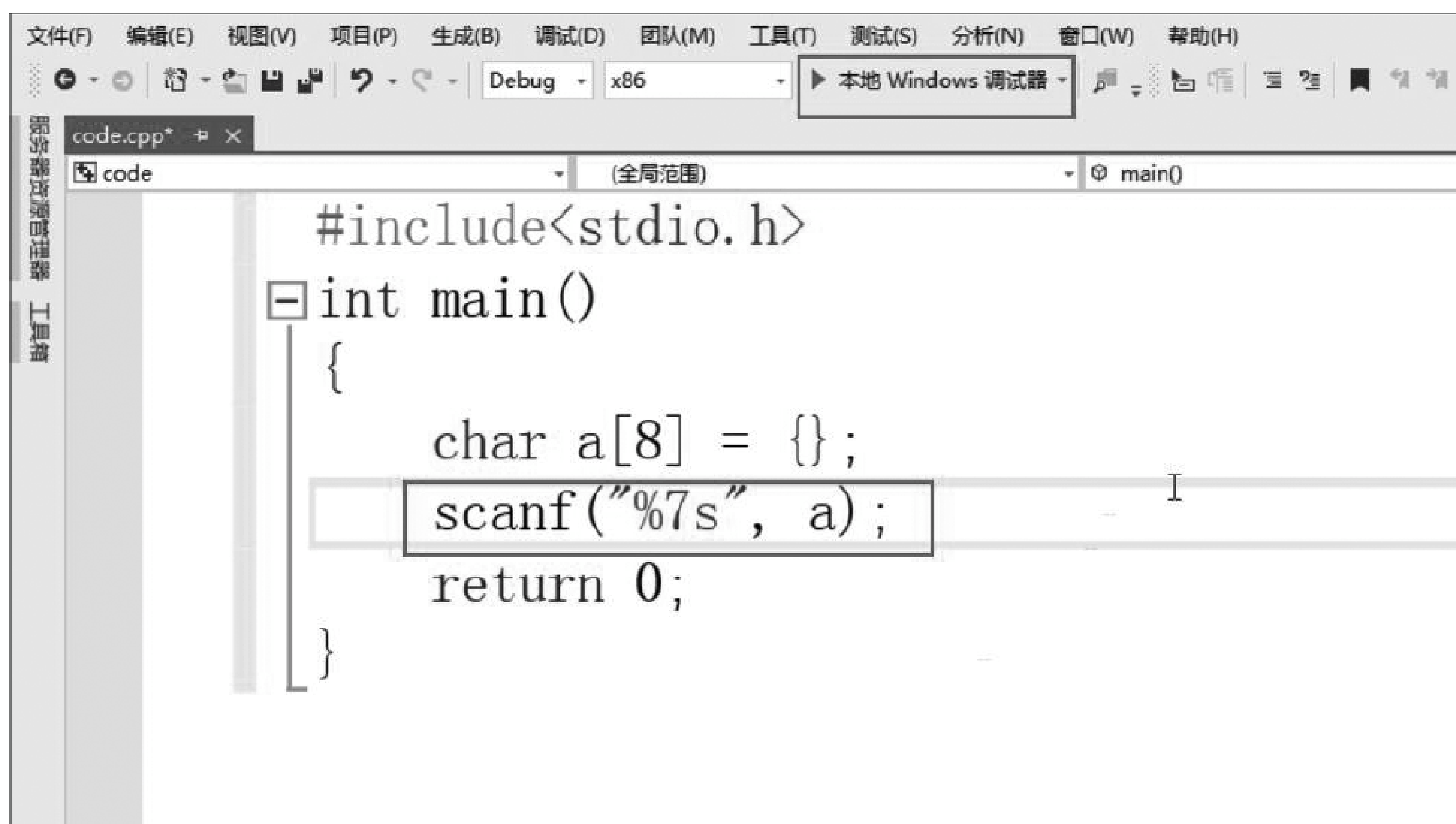


图 2-59　修改代码(2.1.7)

(7) 打开“C:\codemanager”文件夹，删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(9) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入终端机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“发起快速检测”命令。

(12) “任务名称”输入“scanf 没有宽度限制”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-60 所示。



图 2-60　查看结果(2.1.7)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考如果输入超过字符串长度会出现什么情况。

2.1.8　缓冲区下溢缺陷检测实验

【实验目的】

确保所编写的代码中不存在缓冲区下溢的缺陷。

【知识点】

“缓冲区下溢”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个内存处理模块,对一段内存缓冲区进行运算处理,需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:缓冲区下溢。

【实验原理】

缓冲区下溢是一种非常普遍、常危险的漏洞与缺陷,在各种操作系统和应用软件中广

泛存在。缓冲区溢出一般是因为计算机向缓冲区内填充的数据位数超过了缓冲区本身的容量。当一个超长的数据被写入缓冲区时，超出部分被继续写入内存，超出部分的写入会导致程序的异常进行。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-61 所示。

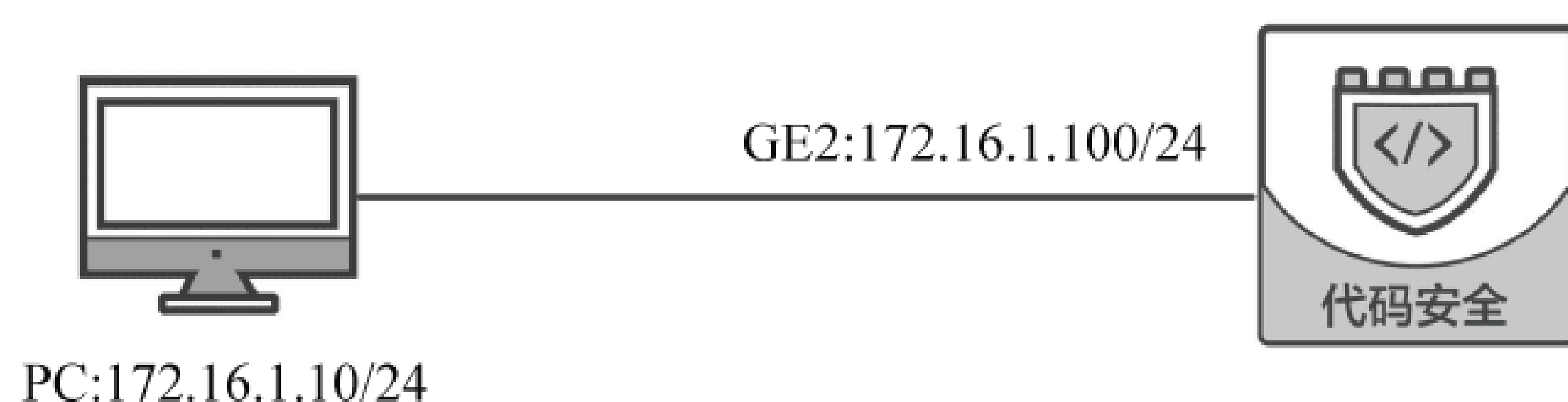


图 2-61　缓冲区下溢缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

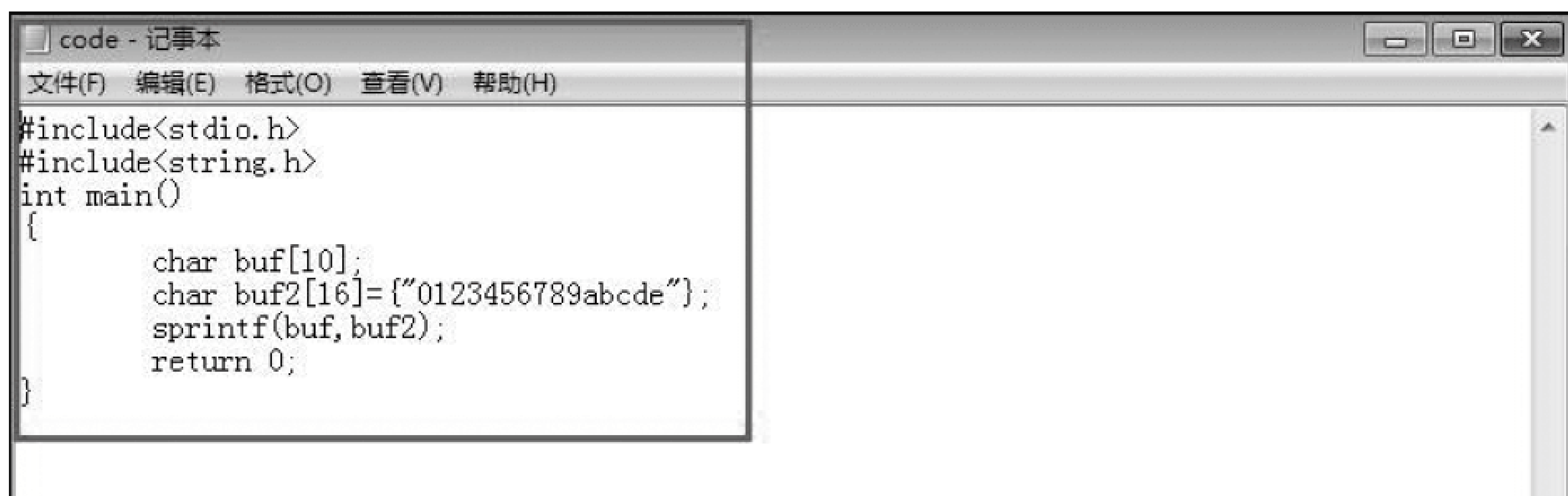
(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

(7) 打开“C:\ ”下的 code.txt，复制其中的代码，如图 2-62 所示。

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资



```
#include<stdio.h>
#include<string.h>
int main()
{
        char buf[10];
        char buf2[16]={"0123456789abcde"};
        sprintf(buf,buf2);
        return 0;
}
```

图 2-62　复制代码(2.1.8)

源管理器”窗口中的 code 项目。

(9) 选择“属性”按钮。

(10) 单击“配置属性”中的“c/c++”下的“预处理器”选项。

(11) 单击“预处理器定义”按钮。

(12) 单击右方的“下拉”按钮。

(13) 单击“编辑”按钮。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框,单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(19) 进入“管理员:VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入终端机“C:\ ”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(22) 在高级界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(23) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(24) 选择“快速检测”→“+发起快速检测”命令。

(25) “任务名称”输入“缓冲区下溢”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(26) 单击“浏览”按钮。

(27) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(28) 选择文件 codemanager,单击“打开”按钮。

(29) 返回“快速检测”界面,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中缓冲区下溢的缺陷。

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中缓冲区下溢的缺陷

（1）在主机终端打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@＃”，单击“登录”按钮。

（2）单击“＋发起快速检测”按钮，查看列表所示检测结果。

（3）找到“缓冲区下溢”，单击右侧的“缺陷审计”按钮。

（4）在左侧查看代码的缺陷，如图 2-63 所示。

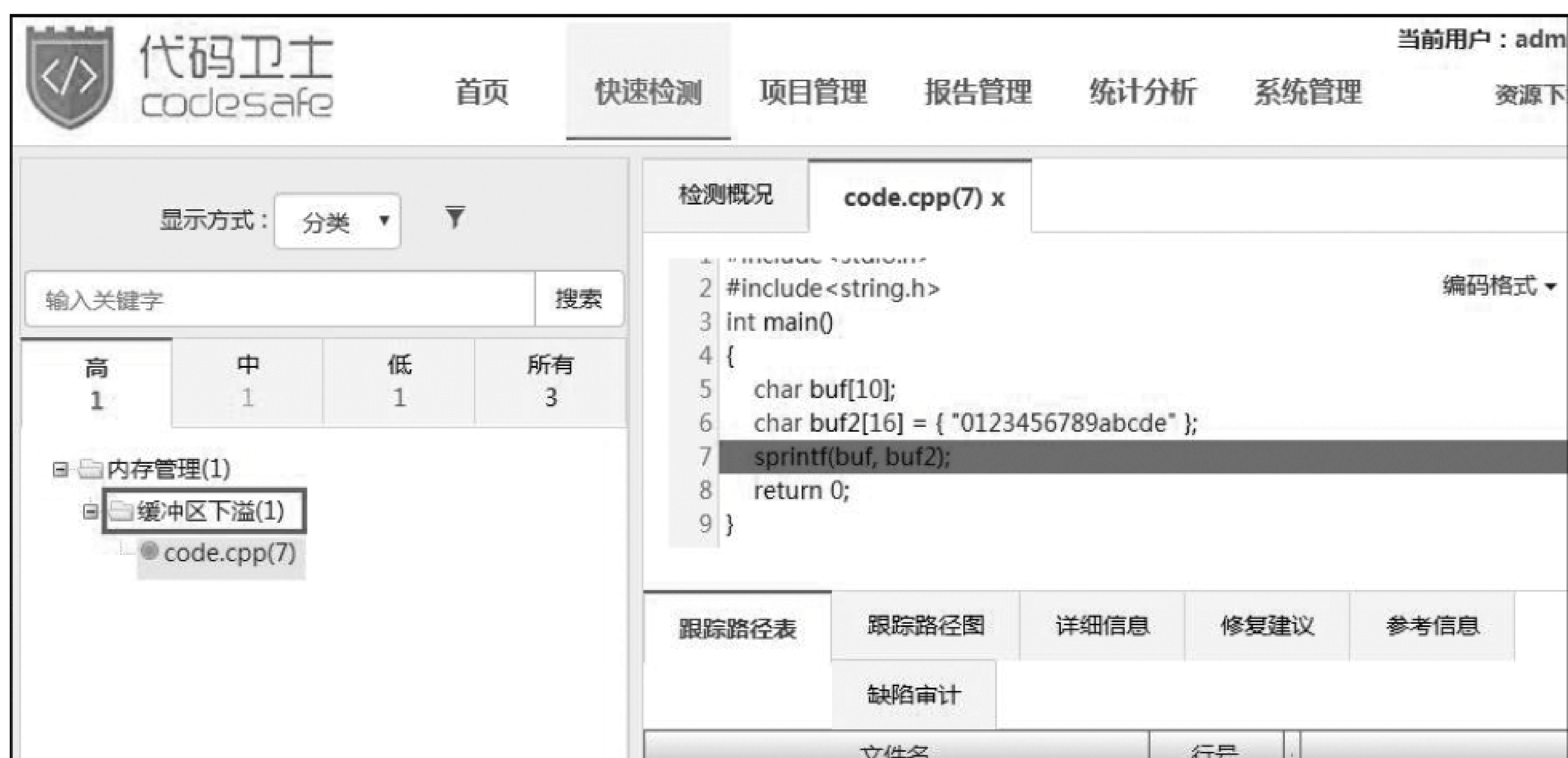


图 2-63 “快速检测”界面(2.1.8)

（5）打开“C:\code”文件夹，双击 code.sln，打开项目。

（6）在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 2-64 所示。

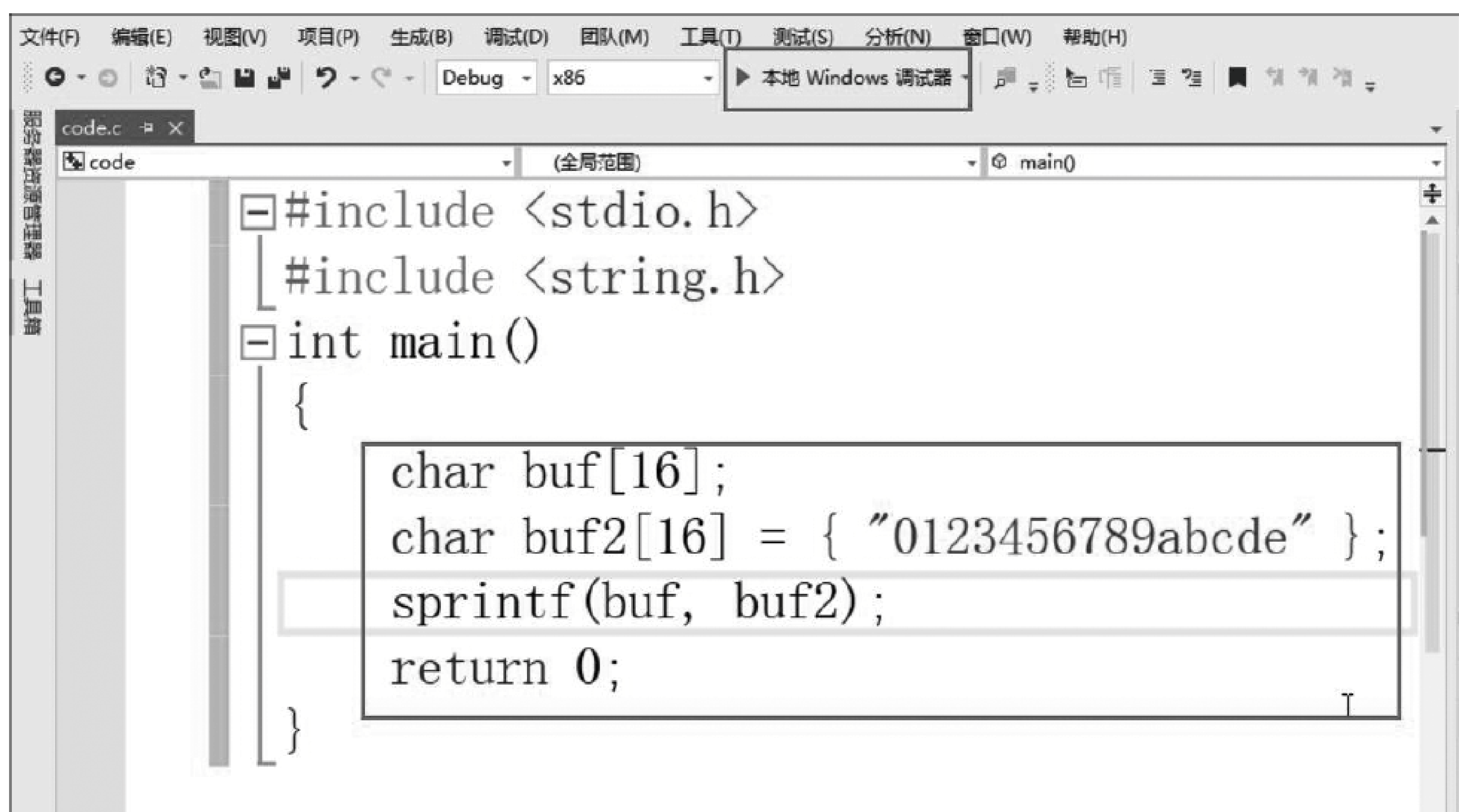


图 2-64 修改代码(2.1.8)

(7) 打开“C:\codemanager”文件夹，删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(9) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入终端机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“缓冲区下溢”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 2-65 所示。



图 2-65　查看结果(2.1.8)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考如何利用缓冲区下溢改变程序进程。

2.1.9　解引用未初始化的指针代码缺陷检测实验

【实验目的】

确保所编写的代码中不存在解引用未初始化的指针的情况。

【知识点】

“解引用未初始化的指针”、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个指针运算模块，对指针及指针指向的内存进行处理，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：解引用未初始化的指针。

【实验原理】

C 语言中指针没有初始化，那么在使用它时，指针可能指向任何地方。解引用一个未初始化的指针则会导致使用一个任意的地址中的内容，导致程序不能正常执行。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-66 所示。

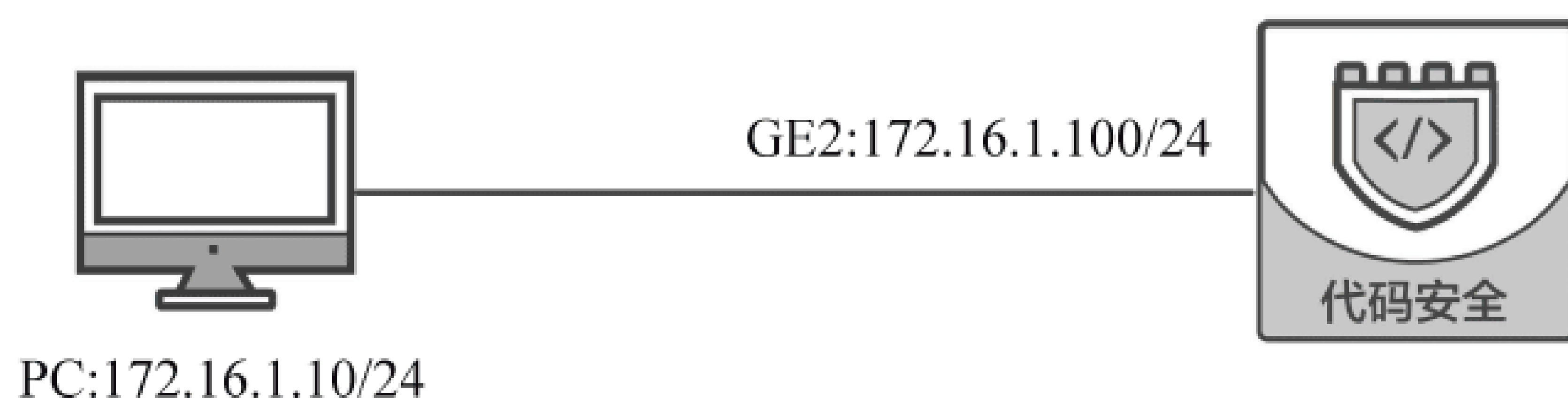


图 2-66　解引用未初始化的指针代码缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015,显示主界面。

(3) 进入主界面,选择“文件”→“新建”→“项目”命令,新建一个项目。

(4) 进入“新建项目”界面,在左侧“模板”选择“Visual C++”命令,界面中间选项选择“空项目”,下侧“名称”选项框内输入 code,单击“位置”右侧的“浏览”按钮,选择“C:\”,其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上“解决方案资源管理器”小窗口,已经新建好 code 项目。右击“源文件”,在弹出的快捷菜单中选择“添加”→“新建项”命令,添加一个新项。

(6) 进入“添加新项”界面,左侧“已安装”默认为“Visual C++”命令,在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code,单击“添加”按钮。

(7) 打开“C:\ ”下的 code.txt,复制其中的代码,如图 2-67 所示。

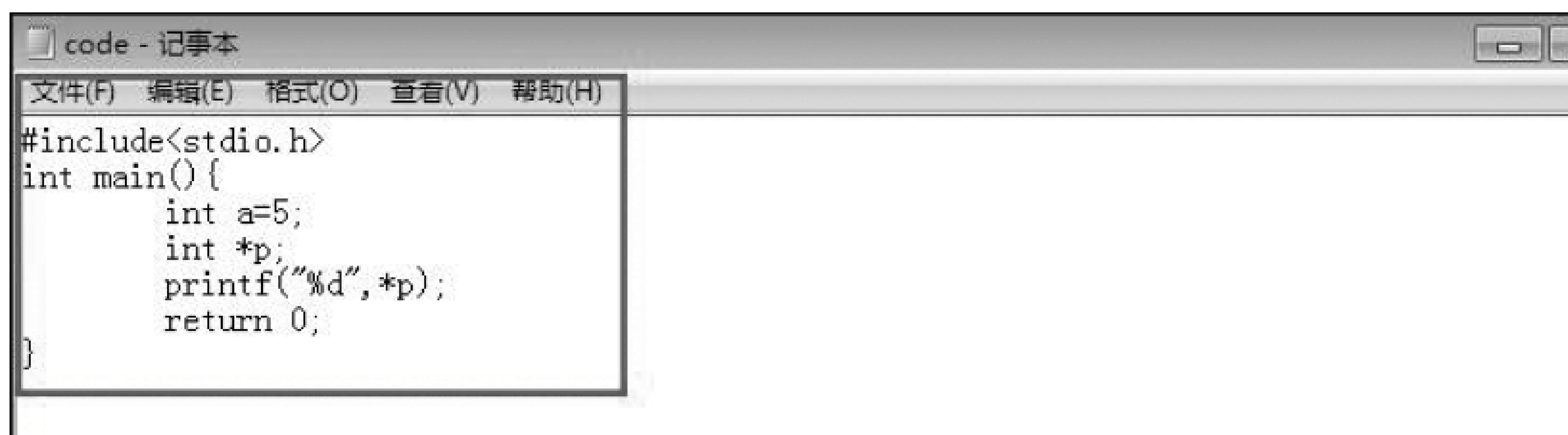


图 2-67　复制代码(2.1.9)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“所有选项”命令,如图 2-68 所示。

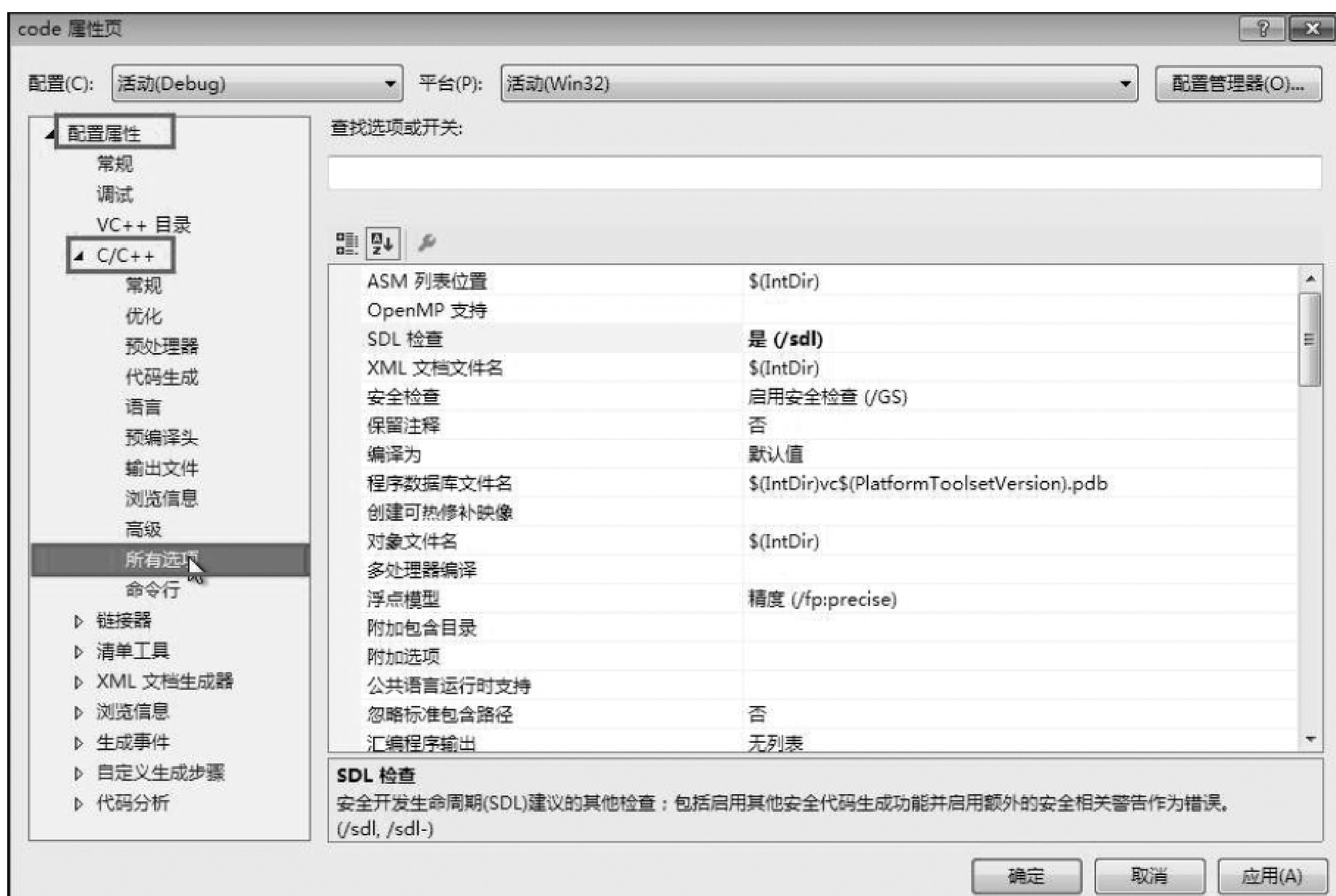


图 2-68　所有选项界面

(11) 选择"SDL 检查"命令，如图 2-69 所示。

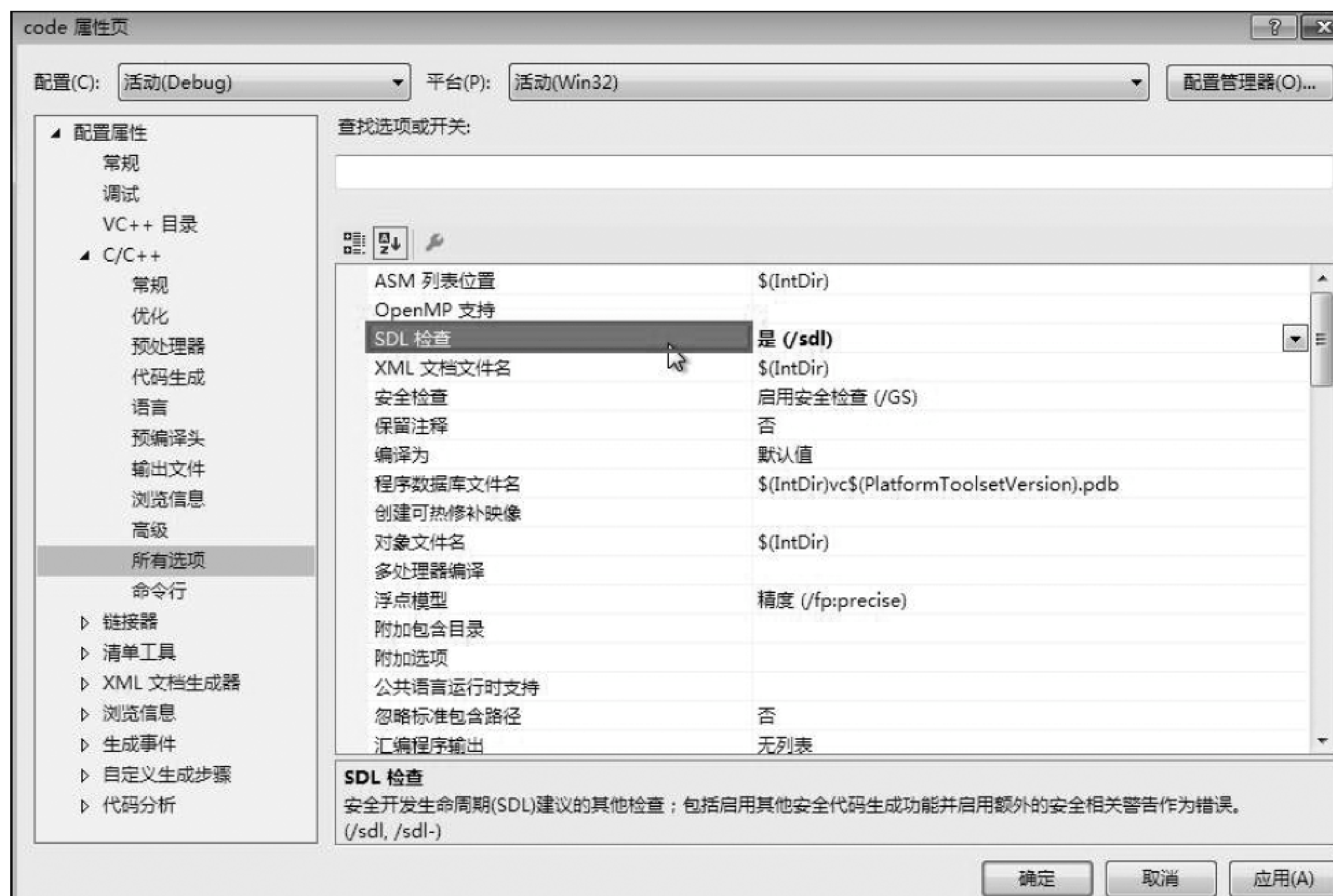


图 2-69　SDL 检查

(12) 单击右方的"下拉"按钮。

(13) 选择"否(/sdl-)"，如图 2-70 所示。

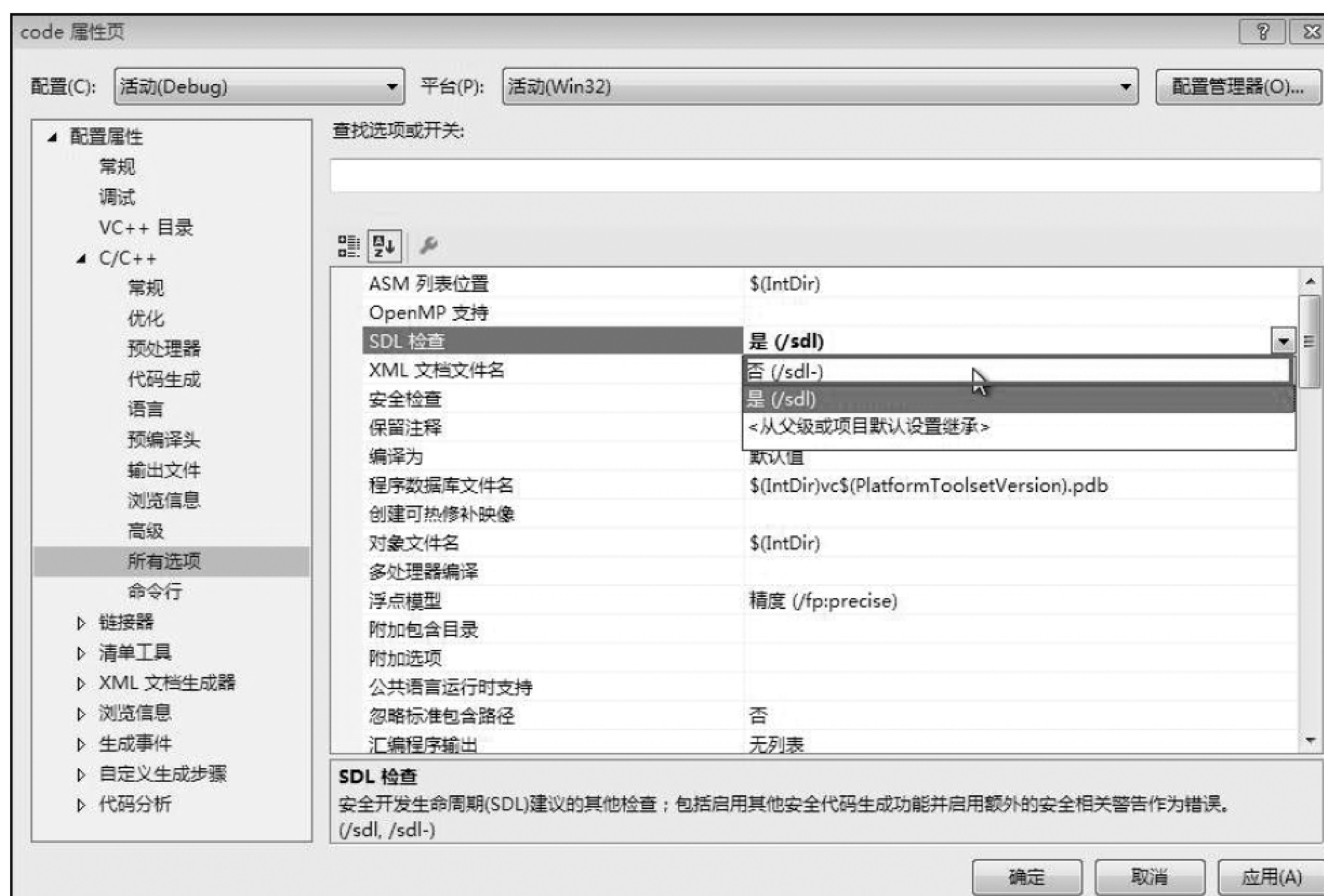


图 2-70　单击"否(/sdl-)"按钮

（14）单击“确定”按钮。

（15）单击“本地 Windows 调试器”按钮。

（16）弹出“编译确认”对话框，单击“是”按钮。

（17）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

（18）进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

（19）进入终端机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

（20）打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中，单击“高级”。

（21）在“高级”界面中单击“继续前往 172.16.1.100(不安全)”。

（22）界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

（23）选择“快速检测”→“＋发起快速检测”命令。

（24）“任务名称”输入“解引用未初始化的指针”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

（25）单击“浏览”按钮。

（26）单击“本地磁盘(C:)”，选择右侧的 codemanager，单击“打开”按钮。

（27）选择文件 codemanager，单击“打开”按钮。

（28）返回“快速检测”界面，单击“发起检测”按钮。

【实验预期】

（1）检测出代码中解引用未初始化的指针的缺陷。

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中解引用未初始化的指针的缺陷

（1）主机终端打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@#”，单击“登录”按钮。

（2）单击“＋发起快速检测”，查看列表所示检测结果。

（3）找到“解引用未初始化的指针”，单击右侧的“缺陷审计”按钮。

（4）在左侧查看代码的缺陷，如图 2-71 所示。

（5）打开“C:\code”文件夹，双击 code.sln，打开项目。

（6）在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 2-72 所示。

（7）打开“C:\codemanager”文件夹，删除其中的所有文件。

（8）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

（9）进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。



图 2-71　“快速检测”界面(2.1.9)

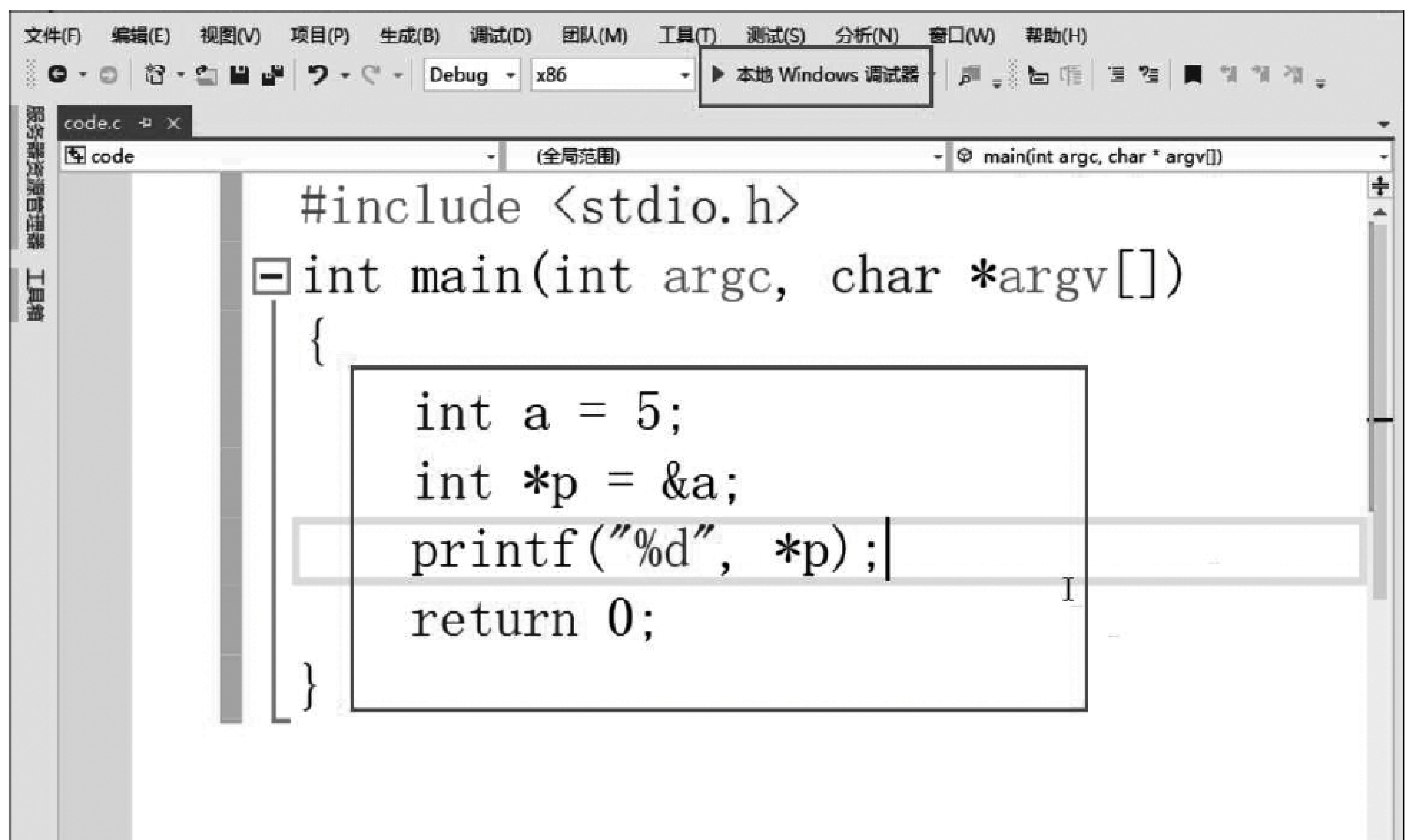


图 2-72　修改代码(2.1.9)

(10) 进入终端机“C:\”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“解引用未初始化的指针”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 2-73 所示。



图 2-73　查看结果(2.1.9)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考解引用未初始化的指针得到的内容的含义。

2.1.10　sizeof 操作符获取数组长度缺陷检测实验

【实验目的】

确保所编写的代码中不存在对指针的 sizeof 操作。

【知识点】

Sizeof 函数对指针的操作的违规、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个数组运算模块，使用 sizeof 对数组长度进行计算，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：获取数组的长度时不要对指针应用 sizeof 操作符。

【实验原理】

在 C 语言中，sizeof() 是一个判断数据类型或者表达式长度的运算符。在编程过程

中，sizeof()可被用来获取数组长度，数组名实际上是一个指向一块连续内存的指针，所以，一个常见的错误是，编程人员定义一个同类型指针，再将数组名指针值赋予这个指针，当用 sizeof()对这个自定义的指针进行操作时，会得到指针本身的长度，如果将该结果作为数组长度进行后续计算，会造成程序运算错误。例如：

```
    int a[4];
    sizeof(a) = 4*4 = 16字节     (int为4个字节)
int *p = a;
    sizeof(p) = 4字节
```

对数组应用 sizeof()，可以得到整个数组分配的字节数(存储全部数据占用的内存字节数)对指针应用 sizeof()，只是得到分配给用来存储一个地址值的指针所用的字节数，即 4 个字节。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-74 所示。

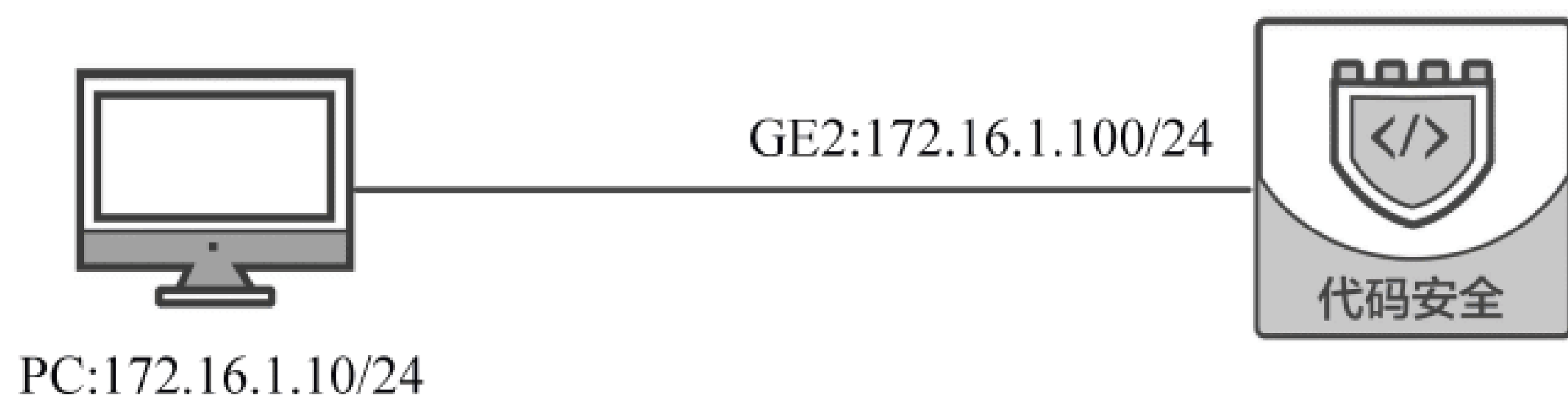


图 2-74　sizeof 操作符获取数组长度缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456，如图 2-75 所示。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”中选择“Visual C++”命令，界面中间选项选

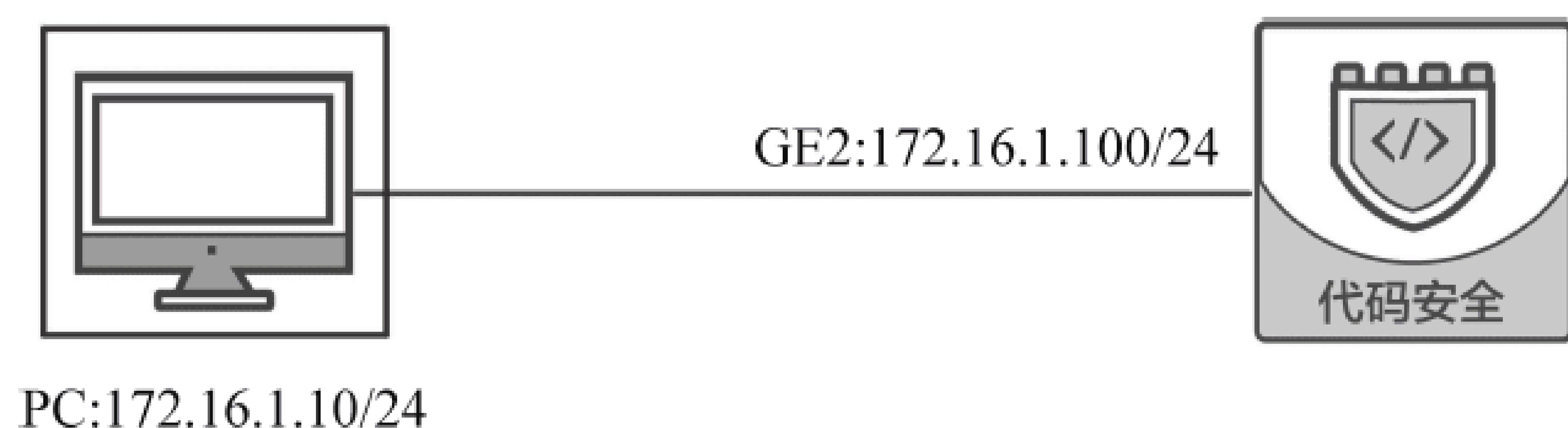


图 2-75　对应实验平台拓扑图

择“空项目”，下侧“名称”选项框内输入 code，单击“位置”选项框右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”按钮，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code，单击“添加”按钮。

(7) 打开“C:\ ”下的 code.txt，复制其中的代码。

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“c/c++”下的“预处理器”选项。

(11) 选择“预处理器定义”命令。

(12) 单击右方“下拉”按钮。

(13) 选择“编辑”命令。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”，单击“确定”按钮。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框，单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(19) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入终端机“C:\”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(22) 在“高级”界面中单击“继续前往 172.16.1.100”按钮。

(23) 界面跳转到代码卫士登录界面，用户名输入 admin，密码输入“admin123!@#”，单击“登录”按钮。

(24) 选择“快速检测”→“+发起快速检测”命令。

(25) “任务名称”输入“获取数组的长度时不要对指针应用 sizeof 操作符”，“开发语

言"选中"C/C++"单选钮,勾选"缺陷检测模板[C/C++]"复选框。

(26) 单击"浏览"按钮。

(27) 单击"本地磁盘(C:)"按钮,选择右侧的 codemanager,单击"打开"按钮。

(28) 选择文件 codemanager,单击"打开"按钮。

(29) 返回"快速检测"界面,等待文件上传成功,单击"发起检测"按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1.检测出代码中释放后使用的缺陷

(1) 在学生本地机打开 Chrome 浏览器,在地址栏中输入"https://172.16.1.100",进入代码安全保障系统登录界面。输入用户名 admin,密码为"admin123!@#",单击"登录"按钮。

(2) 单击"+发起快速检测"按钮,查看列表所示检测结果。

(3) 找到"获取数组的长度时不要对指针应用 sizeof 操作符",单击右侧的"缺陷审计"按钮。

(4) 在左侧查看代码的缺陷,如图 2-76 所示。



图 2-76 "快速检测"界面(2.1.10)

(5) 打开"C:\code"文件夹,双击 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击"本地 Windows 调试器"按钮,运行结束后关闭 VS 2015。

(7) 打开"C:\codemanager 文件夹",删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(9) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”命令。

(12) “任务名称”输入“获取数组的长度时不要对指针应用 sizeof 操作符”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 2-77 所示。



图 2-77　查看结果(2.1.10)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.11　宽窄字符串及其操作函数混淆缺陷检测实验

【实验目的】

确保所编写的代码中不存在宽窄字符串及其操作函数的混淆。

【知识点】

“宽窄字符串操作函数混用”缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个字符串处理模块，调用库函数对字符串进行处理，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要将宽窄字符串及其操作函数混淆。

【实验原理】

宽窄字符与 UTF16，UTF8 不是对应的关系。宽窄字符与一个字符所占的字节数有关，如果一个字符只占一个字节，那么它就是窄字符，一个宽字符通常占 2 个字节。在 c/c++/objective.c 中，如果想把一个窄字符（例如 ASCII 字符）表示为宽字符，通常的做法是使用 wchar 来取代 char，例如：

```
wchar t = 'A';
wchar_t * p = L"Hello!" ;
```

这里每一个字符都占了 2 个字节，并且采用了 UTF16 编码。但 char 类型定义的变量是 UTF8 编码，一个字符占一个字节，这也就造成了 UTF8 可以和 ASCII 编码相兼容，但是 UTF16 却不可以，以致对于 UTF16 的字符串我们必须使用类似于 wprintf 专门的函数来处理宽字符。这就造成了宽窄字符类型分别有对应操作函数的局面，如果互相混用，可能导致程序无法运行出正确的结果。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-78 所示。

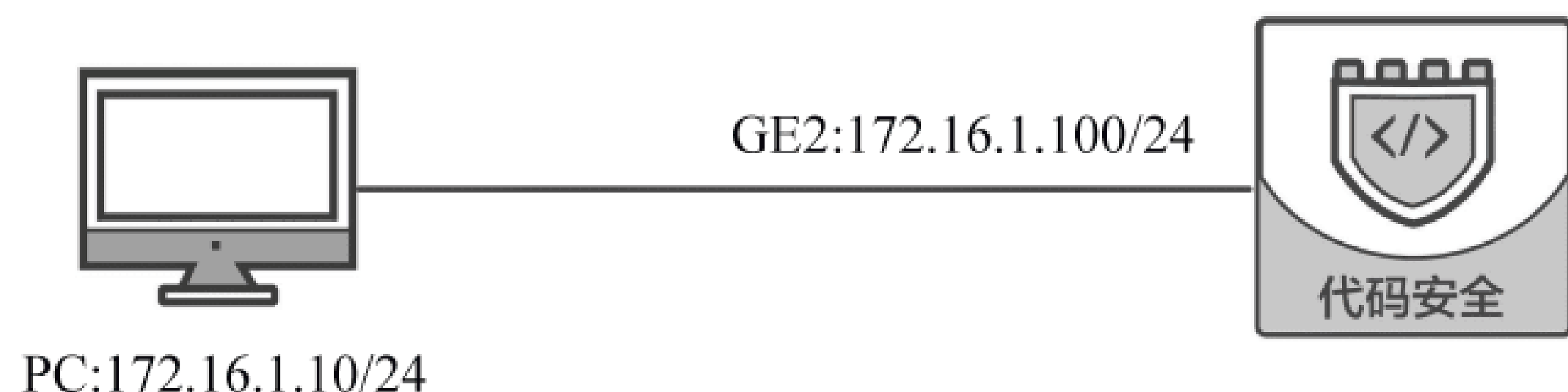


图 2-78　宽窄字符串及其操作函数混淆缺陷检测实验拓扑图

【实验思路】

（1）建立项目文件。

（2）生成代码的中间表示文件。

（3）上传到代码卫士，检测出缺陷。

（4）依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）在终端机桌面双击 Visual Studio 2015，显示主界面。

（3）进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

（4）进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

（5）再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

（6）进入“添加新项”界面，左侧“已安装”默认为“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

（7）打开“C:\ ”下的 code.txt，复制其中的代码，如图 2-79 所示。

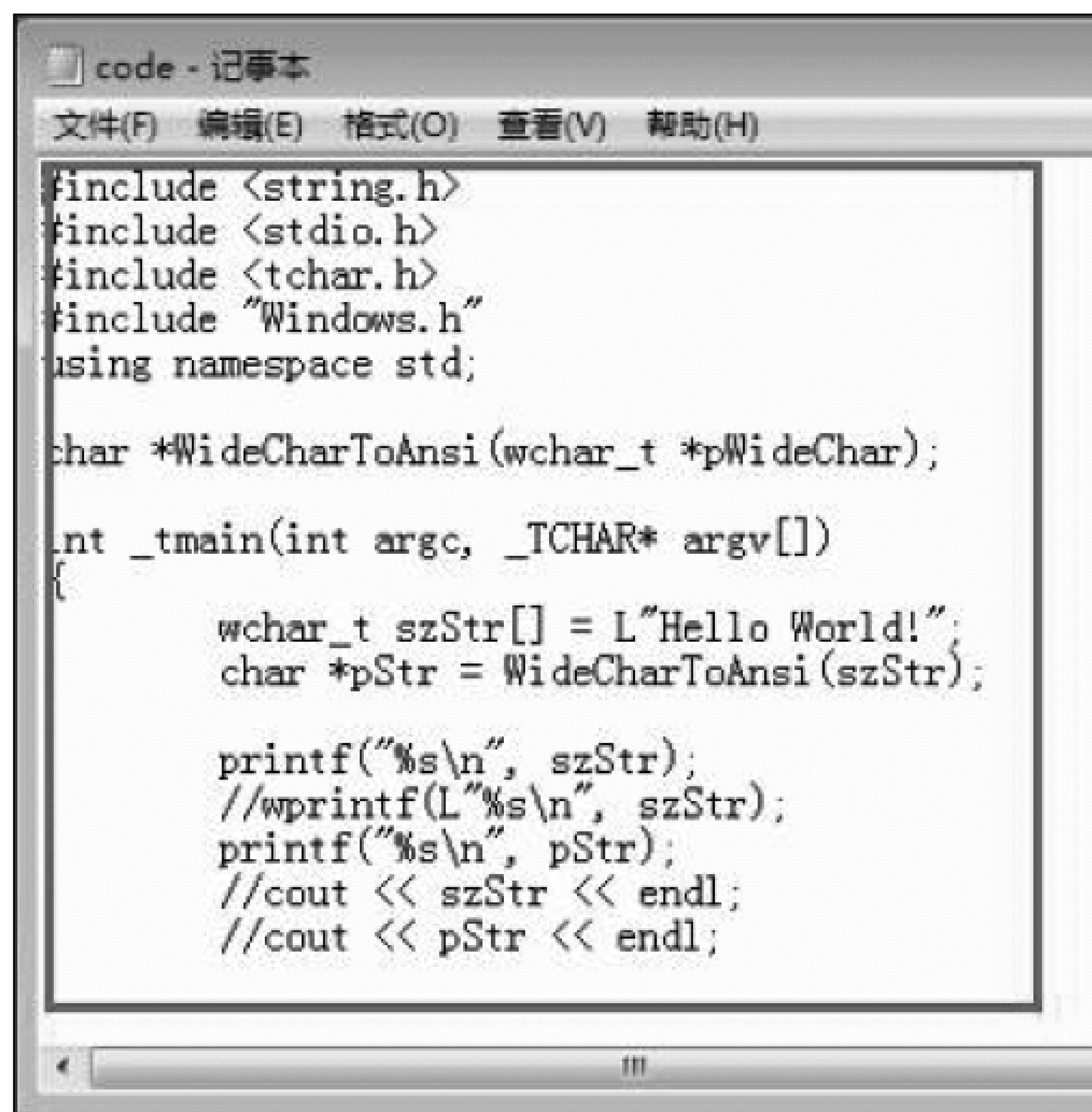


图 2-79　复制代码(2.1.11)

（8）进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

（9）弹出“编译确认”对话框，单击“是”按钮。

（10）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

（11）进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\ ”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”输入“不要将宽窄字符串及其操作函数混淆”,“开发语言”选中“C/C++”单选钮,勾选“缺陷检测模板[C/C++]”复选框。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(20) 选择文件 codemanager,单击“打开”按钮。

(21) 返回“快速检测”界面,等待文件上传成功,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

(1) 在主机终端打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 单击“+发送快速检测”按钮,查看列表所示检测结果。

(3) 找到“不要将宽窄字符串及其操作函数混淆”,单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷,如图 2-80 所示。

(5) 打开“C:\code”文件夹,双击 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,运行结束后关闭 VS 2015,如图 2-81 所示。

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\ ”下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面,选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“不要将宽窄字符串及其操作函数混淆”,“开发语言”选中“C/



图 2-80　“快速检测”界面(2.1.11)

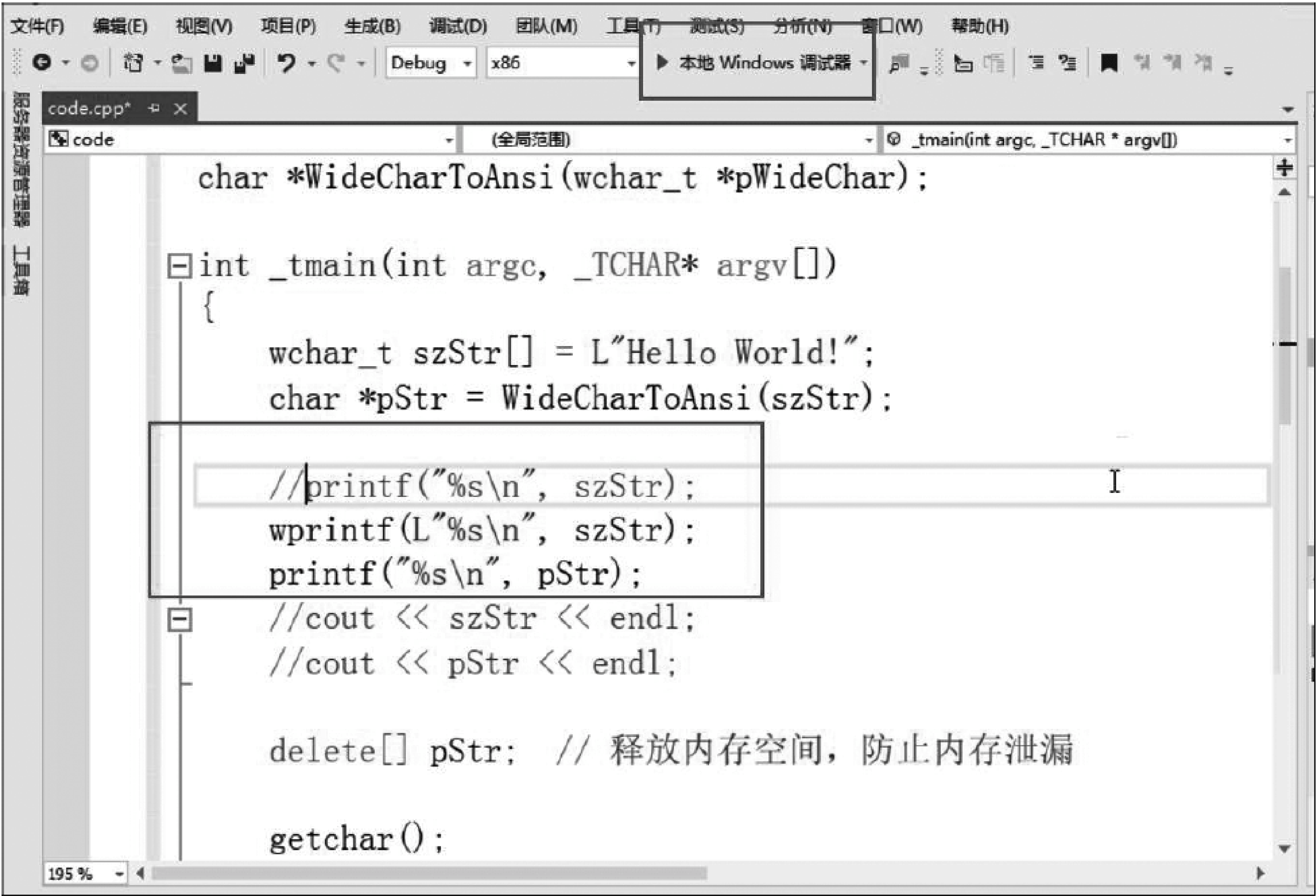


图 2-81　修改代码(2.1.11)

C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(15) 选择文件 codemanager，单击“打开”按钮。

(16) 返回“快速检测”界面，单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

（1）等待任务检测完成后，单击任务右侧的"缺陷审计"按钮。

（2）在左侧查看检测结果，相关问题已经被修改，如图 2-82 所示。



图 2-82　查看结果(2.1.11)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.1.12　代码强制终止执行缺陷检测实验

【实验目的】

确保所编写的代码中不存在强制终止执行的情况。

【知识点】

"强制终止执行"缺陷、C/C++缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个任务调度模块，该模块调用 Windows API 终止线程。需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要强制终止执行。

【实验原理】

很多时候我们需要在程序退出的时候做一些诸如释放资源的操作，但程序退出的方式有很多种，比如 main()函数运行结束、在程序的某个地方用 exit()结束程序、用户通过 Ctrl+C 或 Ctrl+break 操作来终止程序等，因此需要有一种与程序退出方式无关的方法来进行程序退出时的必要处理。方法就是用 exit()调用 atexit()函数来注册程序正常终

止时要被调用的函数。

atexit()函数的参数是一个函数指针，函数指针指向一个没有参数也没有返回值的函数。

```
atexit()的函数原型是：
#include <cstdlib>
int atexit(void(*func)(void));
atexit()成功时返回零，失败时返回非零。
```

函数说明：atexit()用来设置一个程序正常结束前调用的函数。当程序调用 exit()，参数 function 所指定的函数会先被调用，然后才真正由 exit()结束程序。调用 exit()函数终止程序有时候是很危险的，比如导致某段功能代码被跳过从而无法跳转到指定界面或者出现异常终止的情况。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-83 所示。

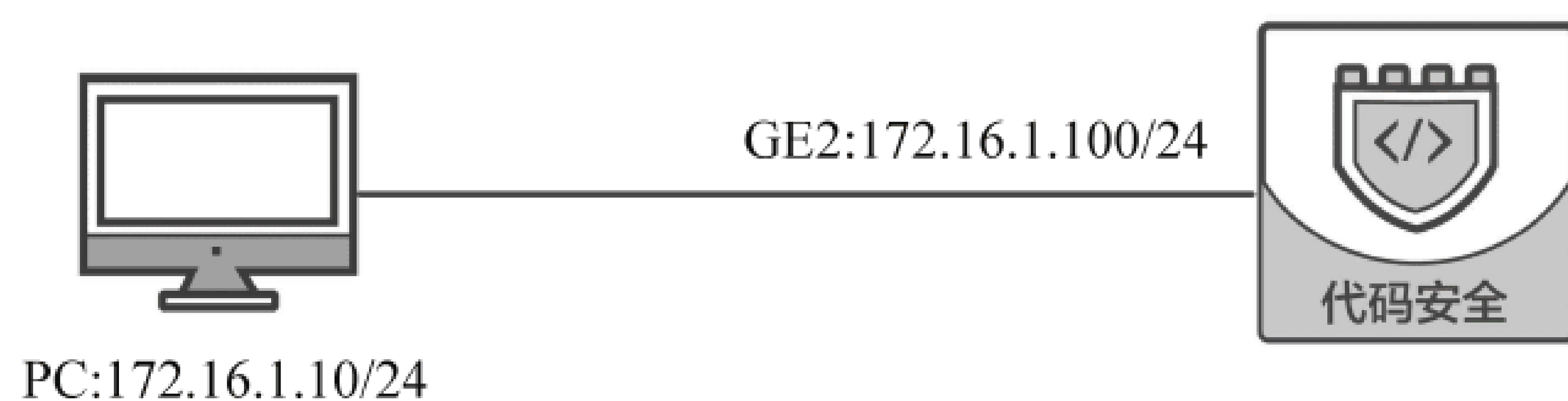


图 2-83　代码强制终止执行缺陷检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择

“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认为“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code，单击“添加”按钮。

(7) 打开“C:\”的 code.txt，复制其中的代码，如图 2-84 所示。

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

(9) 弹出“编译确认”对话框，单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(11) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

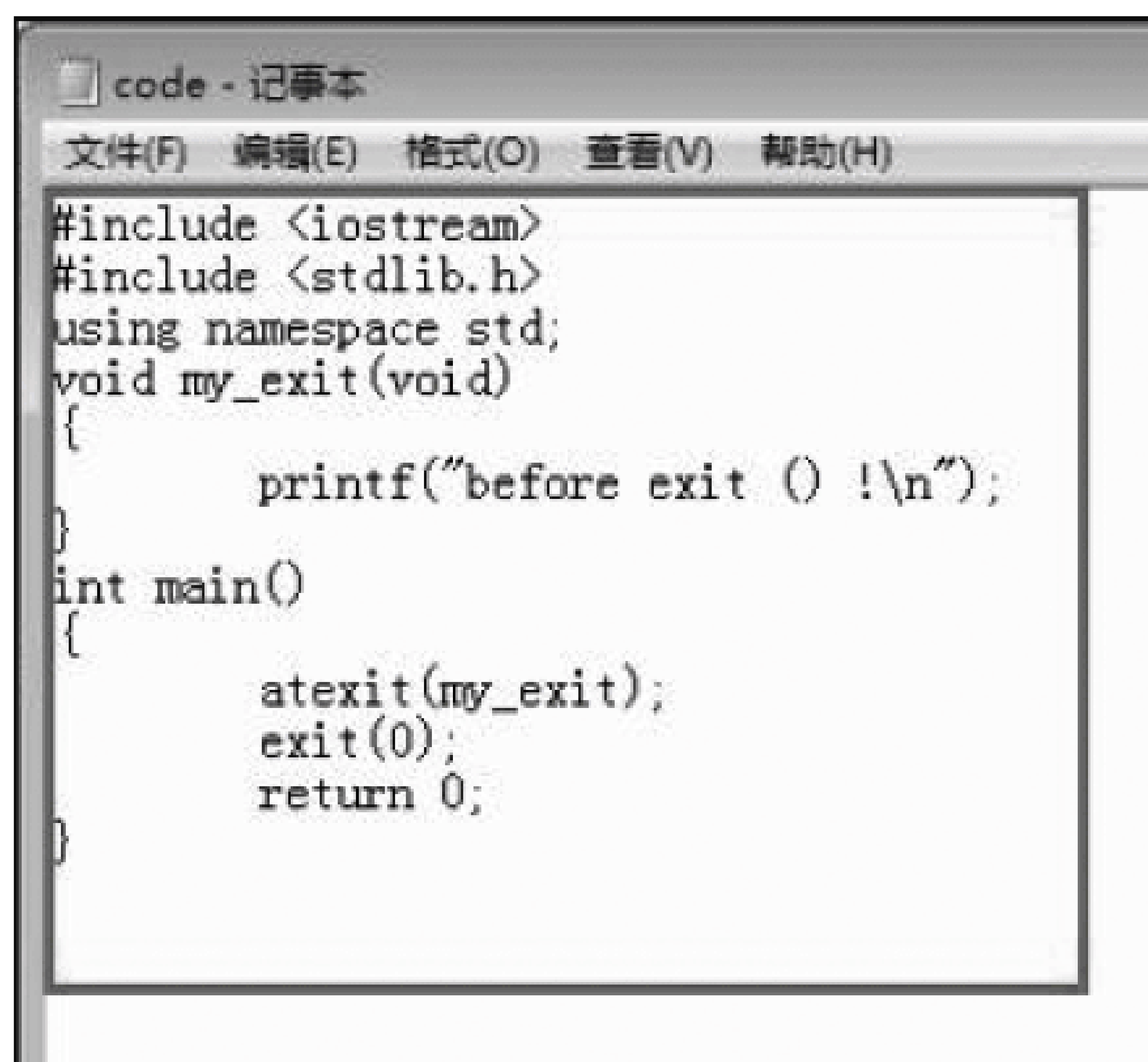


图 2-84　复制代码(2.1.12)

(12) 进入终端机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(13) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”输入“不要强制终止执行”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(18) 单击“浏览”按钮。

(19) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(20) 选择文件 codemanager，单击“打开”按钮。

(21) 返回“快速检测”界面，等待文件上传成功，单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

(1) 在主机终端打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@#”，单击“登录”按钮。

(2) 单击“+发起快速检测”按钮，查看列表所示检测结果。

(3) 找到“不要强制终止执行”，单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷，如图 2-85 所示。



图 2-85　“快速检测”界面(2.1.12)

(5) 打开“C:\code”文件夹，双击 code.sln，打开项目。

(6) 在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，运行结束后关闭 VS 2015，如图 2-86 所示。

(7) 打开“C:\codemanager”文件夹，删除其中的所有文件。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(9) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“不要强制终止执行”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(13) 单击“浏览”按钮。

(14) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

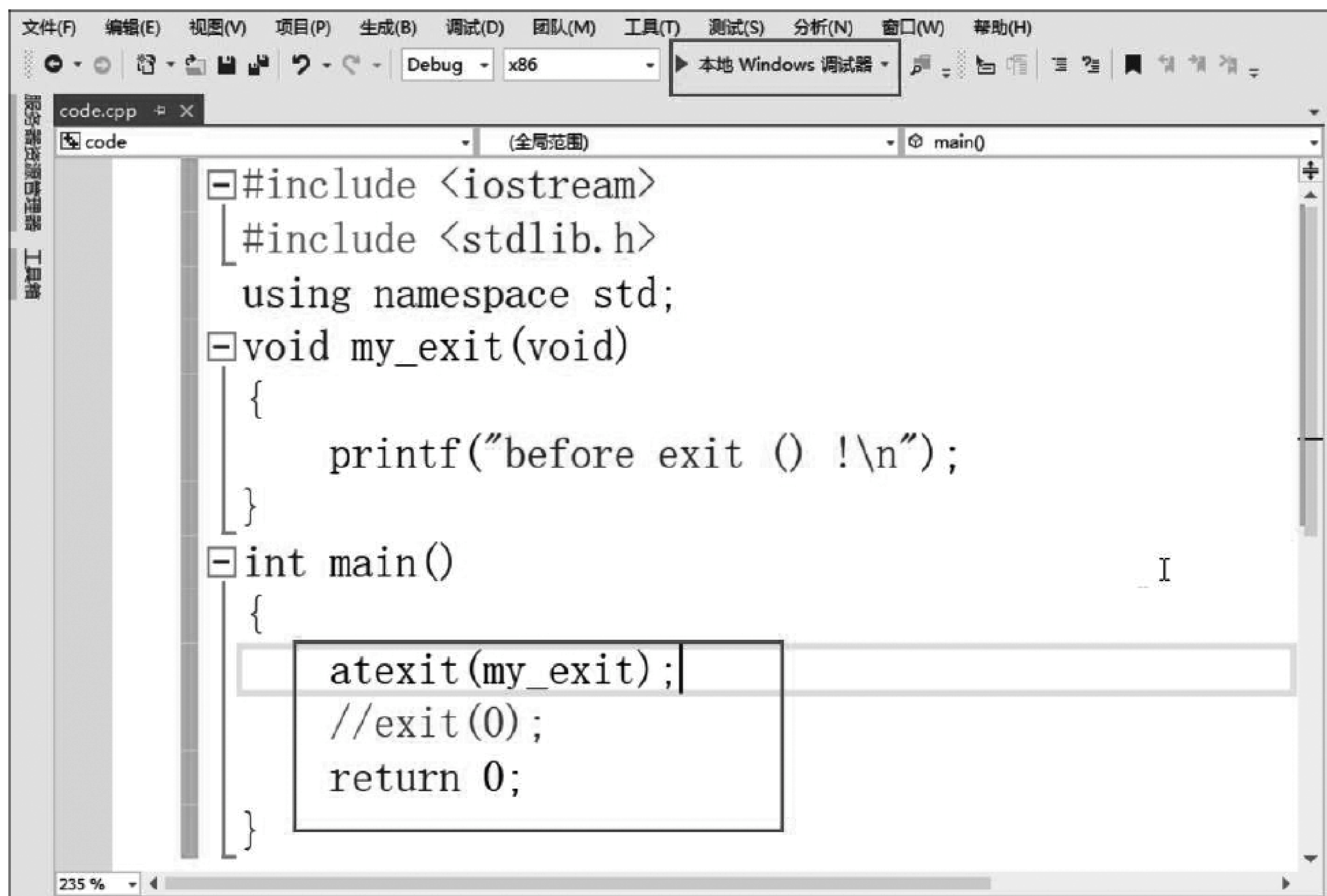


图 2-86　修改代码(2.1.12)

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 给出该缺陷的详细描述和修复建议

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 2-87 所示。



图 2-87　查看结果(2.1.12)

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。

2.2 PHP 缺陷检测

2.2.1　命令注入缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测系统信息处理代码模块是否含有命令注入缺陷,并对发现的命令注入缺陷进行针对性的修复,确保所编写的代码中不含有命令注入缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个系统信息处理代码模块,使用外部命令接口来获取系统数据。需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:命令注入。

【实验原理】

命令注入是指应用程序执行命令的字符串或字符串的一部分来源于不可信赖的数据源,程序没有对这些不可信赖的数据进行验证、过滤,导致程序执行恶意命令的一种攻击方式。

代码安全保障系统能够检测出代码中的缺陷,并给出详细描述以及修复建议,方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-88 所示。

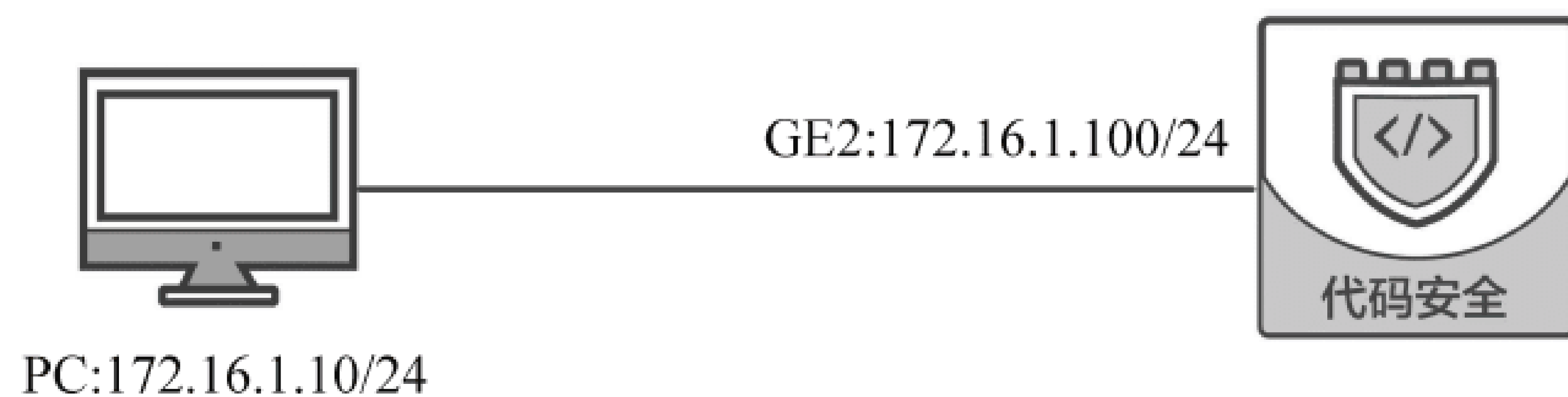


图 2-88　命令注入缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测,检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开“C:\code”目录下的 code.php 文件。

(3) 该代码的功能是使用 GET 方法接受来自用户输入的 IP 地址参数,然后系统执行 ping 命令,从代码中可以看到用户输入的 IP 地址并没有经过过滤或检查,直接拼接到命令中,容易产生命令注入漏洞,如图 2-89 所示。

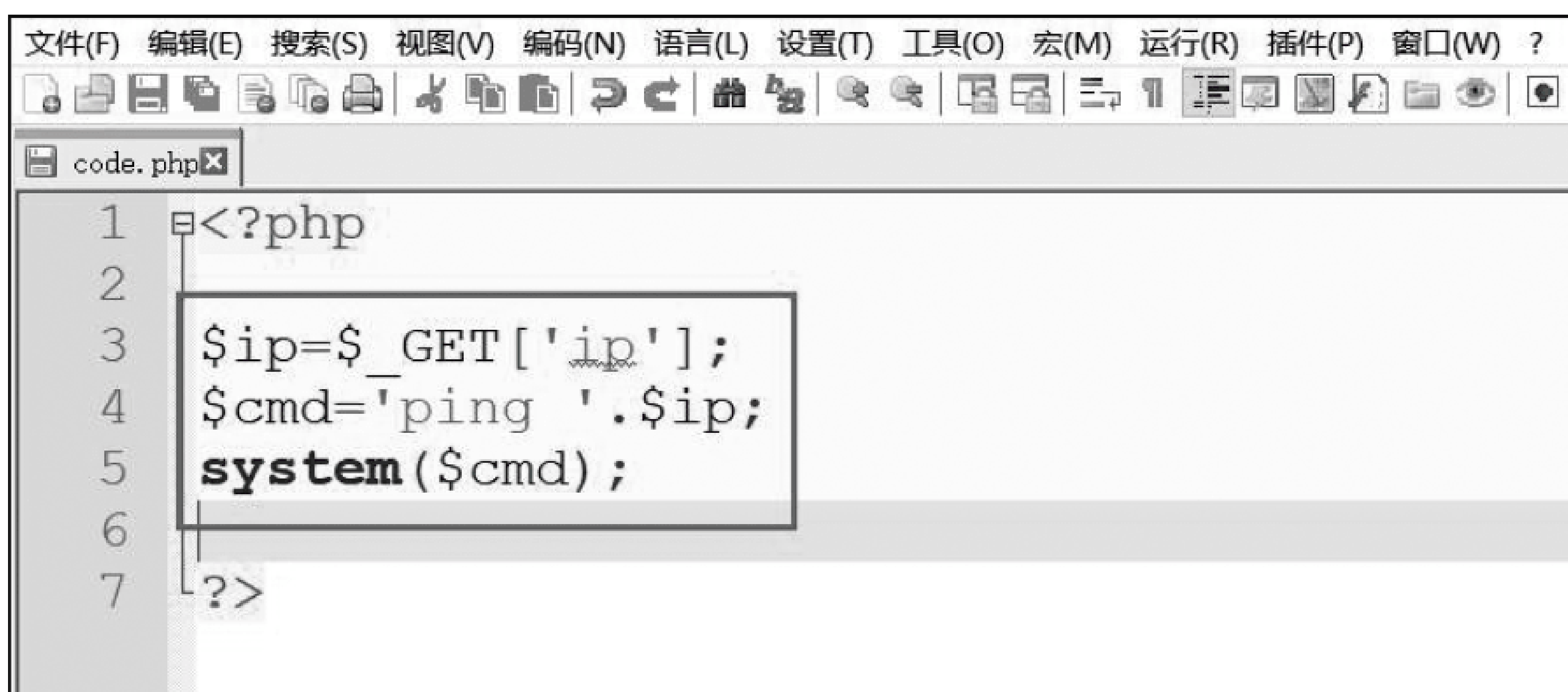


图 2-89　代码解析(2.2.1)

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100(不安全)”按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“命令注入”,“开发语言”选中 PHP 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式,代码卫士平台只能上传“*.zip”文件),其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,选择“缺陷审计”命令(图中最右框位置),如图 2-90 所示。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险等级为高。单击左侧框中的“+”按钮,展开目录,如图 2-91 所示。



图 2-90　开始检测



图 2-91　展开界面(2.2.1)

(11) 选择左侧的"code(5)"命令，可以快速定位有问题的代码，如图 2-92 所示。

(12) 选择"详细信息"命令，可以查看问题代码中该条缺陷的具体信息。信息表明当应用程序执行命令的字符串或字符串的一部分来源于不可信赖的数据源时，若程序不对其进行验证，则容易引发命令注入攻击，如图 2-93 所示。

(13) 选择"修复建议"命令，查看代码卫士给的修复建议。命令注入的防御手段主要有以下两个方法：设置白名单，只允许执行名单中的命令；输入验证，对用户输入的数据进行验证，如图 2-94 所示。

【实验预期】

修改后的代码再次被检测后，缺陷消除。

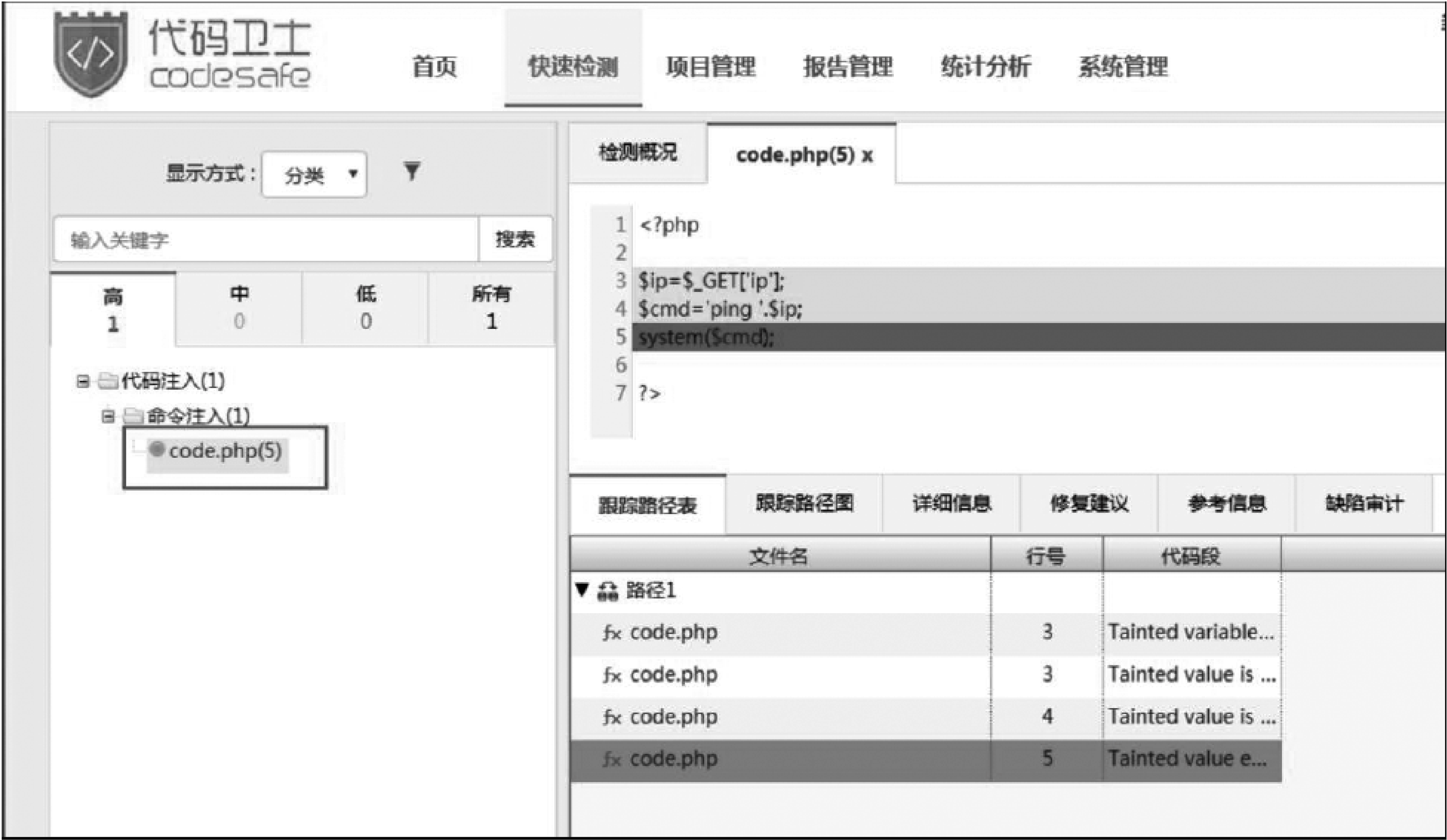


图 2-92 定位有问题的代码(2.2.1)



图 2-93 详细信息(2.2.1)



图 2-94 修复建议(2.2.1)

【实验结果】

（1）最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，使用白名单方式，限制用户输入的 IP 只能在白名单内，保存修改，关闭文件，如图 2-95 所示。

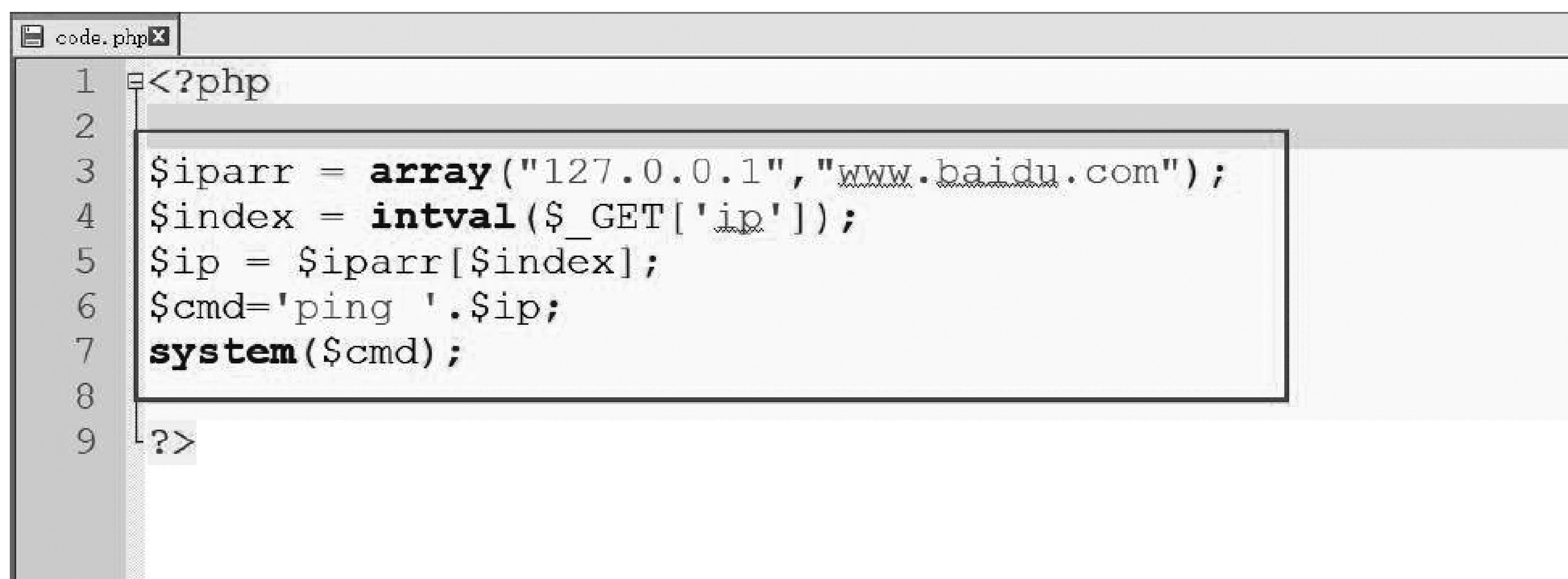


图 2-95　代码修改(2.2.1)

（2）右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

（3）“压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

（4）打开 Chrome 浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

（5）在配置信息界面中，“任务名称”输入“命令注入修复检测”，“开发语言”选中 PHP 单选钮，“源码来源”选择“本地”，“上传文件”选择“C:\code”目录下的“code-fix.zip”文件，其他保持默认配置，单击“发起检测”按钮。

（6）在任务列表界面等待检测完成，可以看到该项目检测完成后，“等级分布”的图形示意为空，缺陷总数为 0，如图 2-96 所示。



图 2-96　检测完成(2.2.1)

【实验思考】

(1) 尝试使用“escapeshellarg()”函数对用户输入的数据进行转义，检测修改后的代码，查看是否可以降低风险等级。

(2) 搭建 Web 环境，利用没有修复的代码尝试复原命令注入漏洞。

2.2.2 SQL 注入缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测图书管理代码模块是否含有 SQL 注入缺陷，并对发现的 SQL 注入缺陷进行针对性的修复，确保所编写的代码不含有 SQL 注入缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个图书管理代码模块，连接数据库对图书数据进行增删改查。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：SQL 注入。

【实验原理】

SQL 注入是通过把 SQL 命令插入提交的 Web 表单或输入域名或界面请求的查询字符串中，从而达到欺骗服务器执行恶意 SQL 命令的漏洞。具体来说，它是利用现有应用程序将恶意的 SQL 命令注入后台数据库并使数据库引擎执行该命令的能力，它可以通过在 Web 表单中输入恶意的 SQL 语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-97 所示。

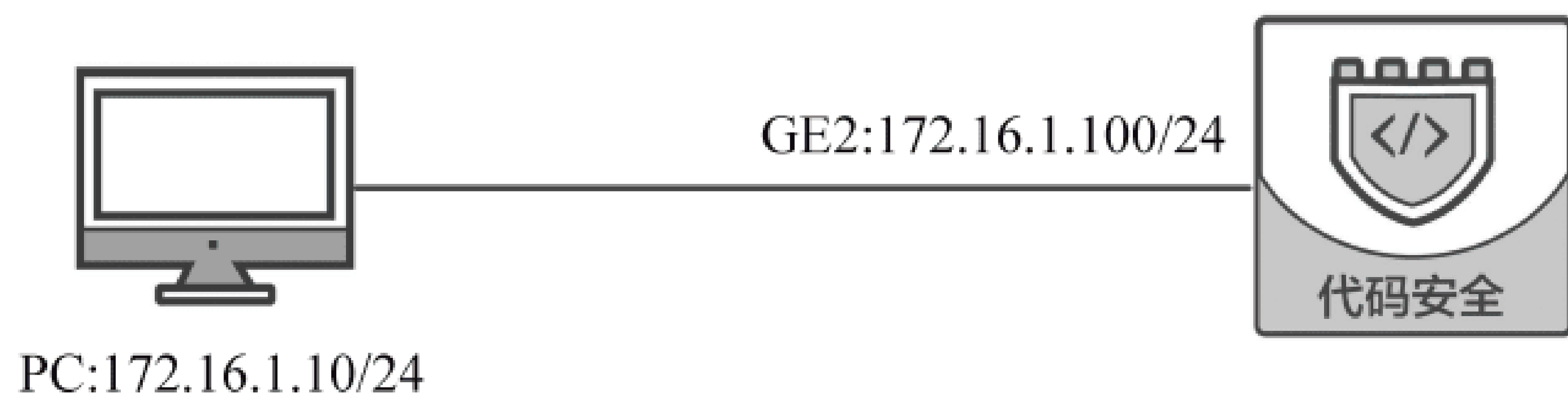


图 2-97 SQL 注入缺陷检测实验拓扑图

【实验思路】

（1）发起 PHP 代码缺陷检测。

（2）根据修改建议对代码进行修改。

（3）对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开“C:\code”目录下的 code.php 文件。

（3）该代码的功能是通过接受来自用户输入的 ID 号，到数据库中查询该 ID 号所对应的用户的“first_name”，从代码中可以看到该程序并未对用户输入的 ID 进行验证或过滤，直接将其放到数据库中进行查询，极易引发 SQL 注入，如图 2-98 所示。

```
code.php
1  <?php
2
3  if( isset( $_REQUEST[ 'Submit' ] ) ) {
4      // Get input
5      $id = $_REQUEST[ 'id' ];
6
7      // Check database
8      $query  = "SELECT first_name FROM users WHERE user_id = '$id';";
9      $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<
10
11     // Get results
12     while( $row = mysqli_fetch_assoc( $result ) ) {
13         // Get values
14         $first = $row["first_name"];
15
16         // Feedback for end user
17         $html .= "<pre>ID: {$id}<br />First name: {$first}</pre>";
18     }
19
20     mysqli_close($GLOBALS["___mysqli_ston"]);
21 }
```

图 2-98　代码解析(2.2.2)

（4）打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

（5）单击“继续前往 172.16.1.100(不安全)”按钮。

（6）进入代码卫士登录界面。输入用户名和密码（默认用户名为 admin，密码为“admin123!@＃”），单击“登录”按钮。

（7）进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

（8）在配置信息界面中，“任务名称”输入“SQL 注入”，“开发语言”选中 PHP 单选钮，单击“上传文件”处右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件（此文件是 code.php 文件的压缩包格式，代码卫士平台只能上传“＊.zip”文件），其他保持默认配置，单击“发起检测”按钮。

（9）发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时，单击“缺陷审计”按钮。

（10）可以看到代码卫士检测出了相关缺陷，并判断该缺陷的风险等级为低。单击左侧框中的“＋”按钮展开目录，如图 2-99 所示。



图 2-99　展开界面(2.2.2)

（11）选择左侧的“code.php(9)”命令，可以快速定位有问题的代码。

（12）选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。详细信息中举例说明了 SQL 注入产生的原因及攻击方法，如图 2-100 所示。

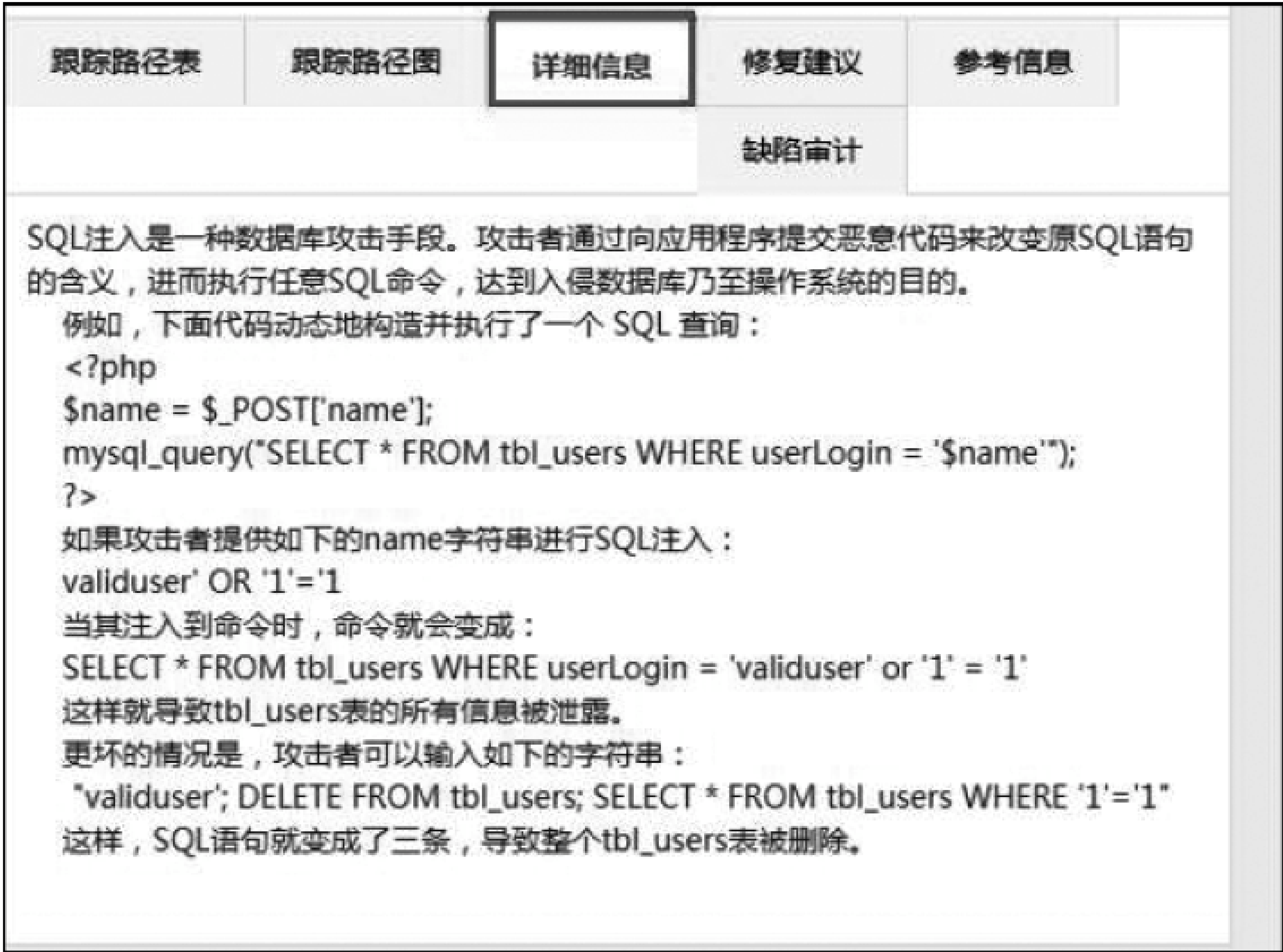


图 2-100　详细信息(2.2.2)

（13）选择“修复建议”命令，查看代码卫士给的修复建议。防止 SQL 注入的方法主要有 3 个：对用户的输入进行合理验证；使用预编译 SQL 语句，使用参数化 SQL 查询的

方法；创建一份合法的字符串列表，数据库仅可执行该表中的命令。

【实验预期】

修改后的代码再次被检测后缺陷消除。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，代码中限制返回的查询结果数量为 1 时，才会输出，防止了大量数据的泄露，同时采用了 PDO 预编译技术，隔离了代码与数据，使得用户输入的数据不再被当作代码执行，采用这种方案，杜绝了 SQL 注入漏洞的发生，相关代码如图 2-101 所示，保存修改，关闭文件。

```
 7  // Get input
 8  $id = $_GET[ 'id' ];
 9
10  // Was a number entered?
11  if(is_numeric( $id )) {
12      // Check the database
13      $data = $db->prepare( 'SELECT first_name FROM users WHERE user_id = (:id) LIMIT 1;' );
14      $data->bindParam( ':id', $id, PDO::PARAM_INT );
15      $data->execute();
16      $row = $data->fetch();
17
18      // Make sure only 1 result is returned
19      if( $data->rowCount() == 1 ) {
20          // Get values
21          $first = $row[ 'first_name' ];
22
23          // Feedback for end user
24          $html .= "<pre>ID: {$id}<br />First name: {$first}</pre>";
25      }
26  }
27
28
29  Generate Anti-CSRF token
```

图 2-101　代码修改(2.2.2)

(2) 右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开 Chrome 浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“SQL 注入修复检测”，“开发语言”选中 PHP 单选钮，“源码来源”设置为“本地”，“上传文件”选择“C:\code”目录下的“code-fix.zip”文件，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“等级分布”的图形示意为空，缺陷总数为 0，如图 2-102 所示。

【实验思考】

(1) 编写代码尝试对用户的输入进行验证，只允许用户输入数字。

(2) 编写代码对进入数据库的特殊字符(如“'”“\”“＜”“＞”“&”“*”和“;”等)进行转义处理。



图 2-102　检测完成(2.2.2)

2.2.3　存储型 XSS 缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测文章管理代码模块是否有存储型 XSS 缺陷，并对发现的存储型 XSS 缺陷进行针对性的修复，确保所编写的代码中不存在存储型 XSS 缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个文章管理代码模块，从浏览器读取需要显示的分类名称，并展示在界面上。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：存储型 XSS。

【实验原理】

存储型 XSS 漏洞又称为持久性 XSS 漏洞，是指程序在没有对用户输入的数据进行验证或过滤的情况下直接将其保存在文件或数据库中，同时该不可信的数据从存储中被获取后，在没有对其进行编码或转义的情况下直接返回给用户。这种不安全的程序容易使用户遭受存储型 XSS 攻击。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP11 台。

【实验拓扑】

实验拓扑如图 2-103 所示。

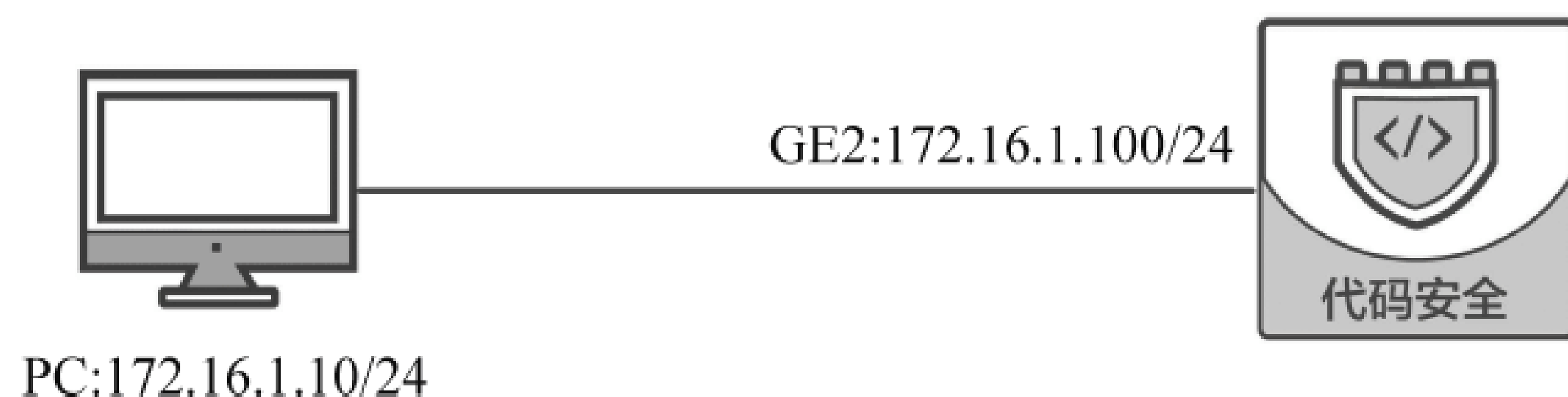


图 2-103　存储型 XSS 缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

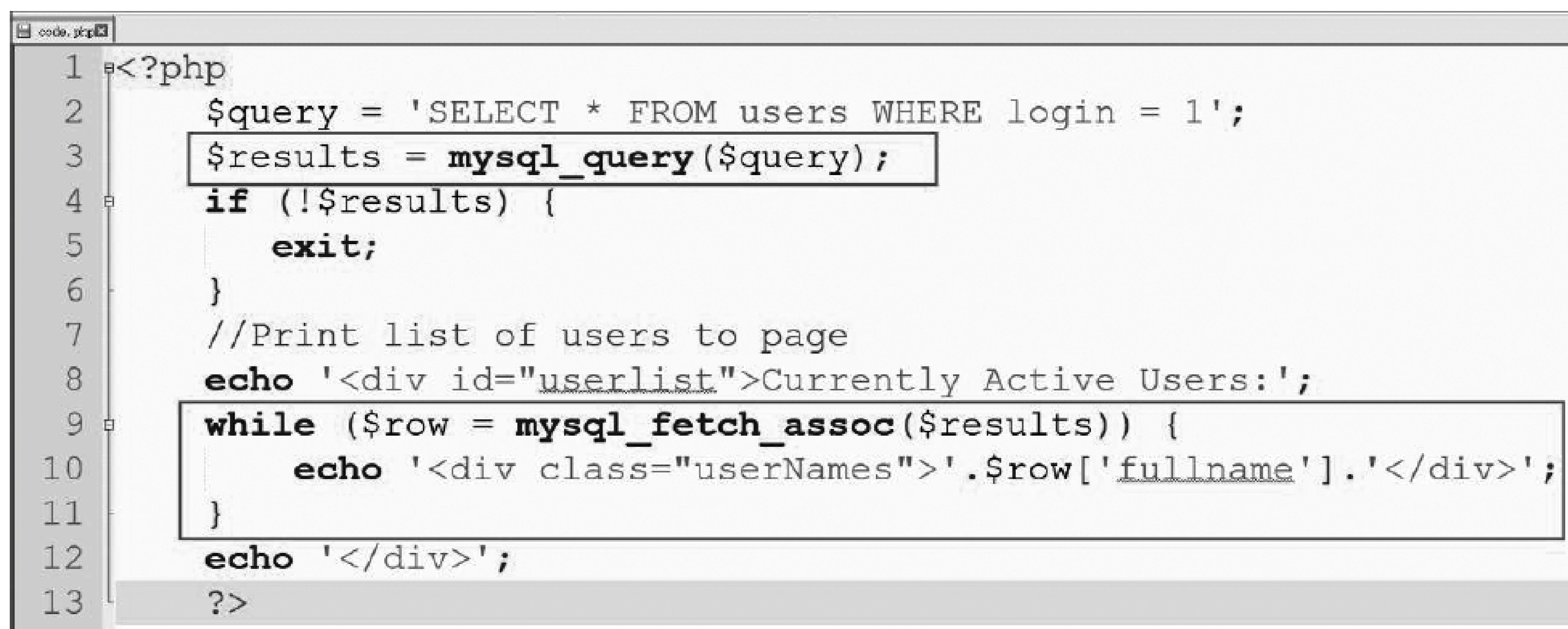
(3) 对修改后的 PHP 代码发起缺陷检测,检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,则输入 123456。

(2) 打开"C:\code"目录下的 code.php 文件。

(3) 该程序的功能是浏览器读取存储在数据库中的文章,不经过滤或编码直接显示在用户的浏览器中。若数据库存储的文章中含有恶意代码,则用户容易遭受存储型 XSS 攻击,如图 2-104 所示。



```
1  <?php
2      $query = 'SELECT * FROM users WHERE login = 1';
3      $results = mysql_query($query);
4      if (!$results) {
5          exit;
6      }
7      //Print list of users to page
8      echo '<div id="userlist">Currently Active Users:';
9      while ($row = mysql_fetch_assoc($results)) {
10         echo '<div class="userNames">'.$row['fullname'].'</div>';
11     }
12     echo '</div>';
13     ?>
```

图 2-104　代码解析(2.2.3)

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(5) 单击"继续前往 172.16.1.100(不安全)"按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(7) 进入代码卫士后,选择"快速检测"→"+发起快速检测"命令。

(8) 在配置信息界面中,“任务名称”输入“存储型 XSS”,“开发语言”选中 PHP 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式,代码卫士平台只能上传“*.zip”文件),其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,选择最右侧的“缺陷审计”命令。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险等级为高。单击左侧框中的“+”按钮,展开目录,如图 2-105 所示。



图 2-105 展开界面(2.2.3)

(11) 选择左侧的“code.php(10)”命令,可以快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。信息表明应用程序从数据库或其他后端数据存储获取不可信赖的数据,在未检验数据是否存在恶意代码的情况下,便将其传递给了 Web 用户,应用程序将易于受到存储型 XSS 攻击,如图 2-106 所示。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。代码卫士提供了三种预防 XSS 攻击的方法:对用户的输入进行合理验证;对所有不可信赖的数据进行恰当的输出编码;为避免攻击者利用跨站脚本漏洞进行 Cookie 劫持攻击,应该为 Cookie 设置 HTTPOnly 属性。

【实验预期】

修改后的代码再次被检测后等级降低。

【实验结果】

(1) 最小化浏览器窗口,打开“C:\code”目录下的 code.php 文件,根据修复建议对代码进行修改,输出时对数据使用“htmlspecialchars()”函数进行 HTML 编码(相关代码改



图 2-106　详细信息(2.2.3)

为"htmlspecialchars($ row['papername'])"，保存修改，关闭文件，如图 2-107 所示。

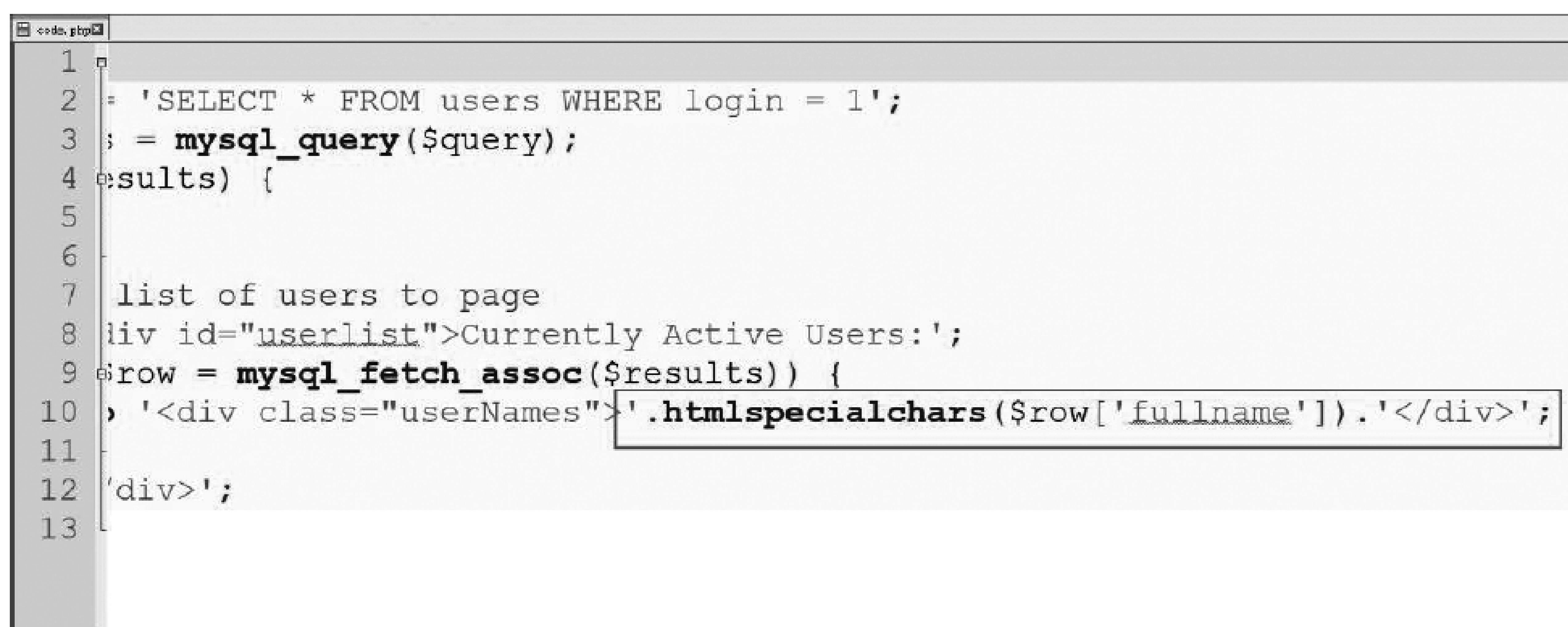


图 2-107　代码修改(2.2.3)

(2) 右击 code.php，在弹出的快捷菜单中选择"添加到压缩文件(A)..."命令。

(3) "压缩到"中的文件名称输入"code-fix.zip"，单击"立即压缩"按钮。

(4) 打开 Chrome 浏览器，在之前的界面上单击"快速检测"按钮。

(5) 在配置信息界面中，"任务名称"输入"存储型 XSS 修复检测"，"开发语言"选中 PHP 单选钮，"源码来源"设置为"本地"，"上传文件"设置为"C:\code"目录下的"code-fix.zip"文件，其他保持默认配置，单击"发起检测"按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，"等级分布"的图形示意为中，风险等级已经降低，如图 2-108 所示。

(7) 可以看到风险等级已经降为"中"，如图 2-109 所示。

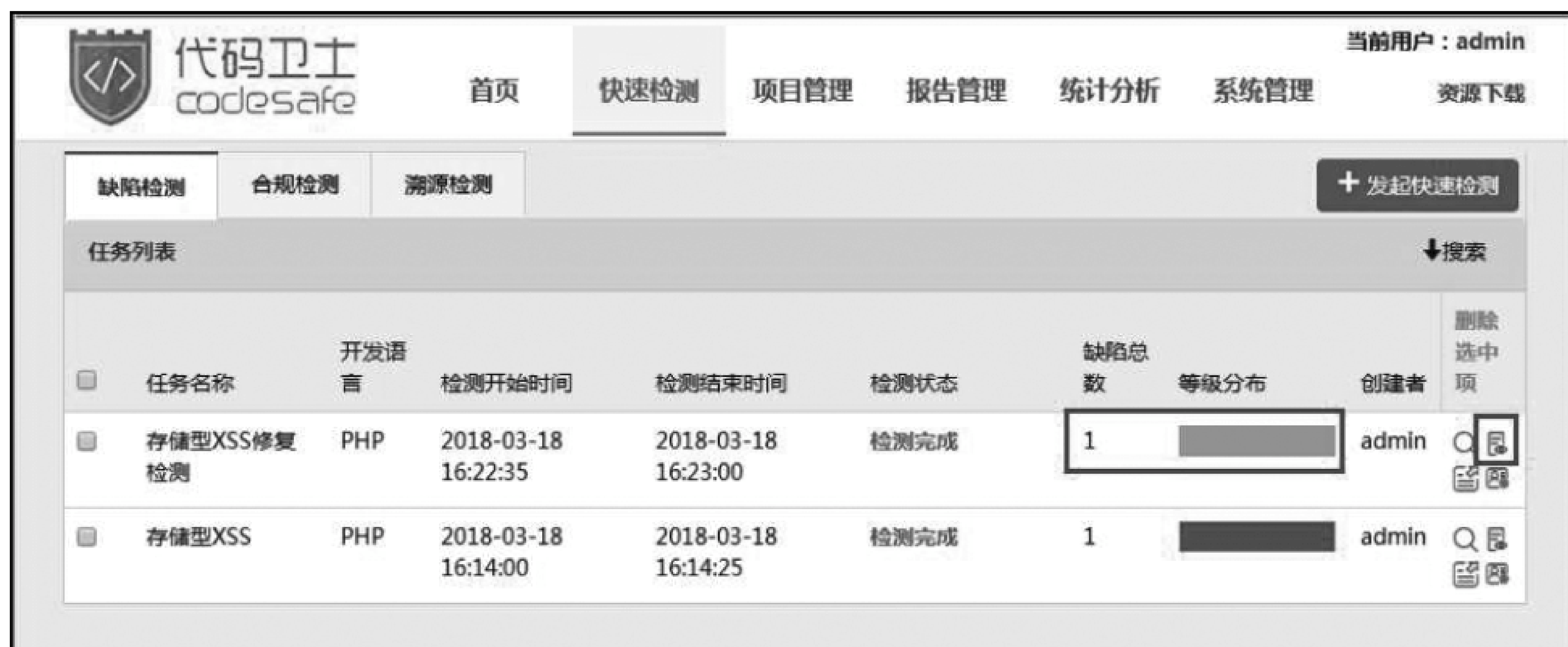


图 2-108　检测完成(2.2.3)



图 2-109　风险等级降低

【实验思考】

(1) 使用 htmlspecialchars()函数的作用是什么？

(2) 如何为 Cookie 设置 HTTPOnly 属性？

2.2.4　反射型 XSS 缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测文章管理代码模块是否有反射性 XSS 缺陷，并对发现的反射性 XSS 缺陷进行针对性的修复，确保所编写的代码中不存在反射性 XSS 缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个文章管理代码模块，从浏览器读取需要显示的分类名称，并展示在界面上。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：反射型 XSS。

【实验原理】

应用程序通过 Web 请求获取不可信赖的数据，在未检验数据是否存在恶意代码的情况下，便将其传送给 Web 用户，用户很容易受到反射型 XSS 攻击。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-110 所示。

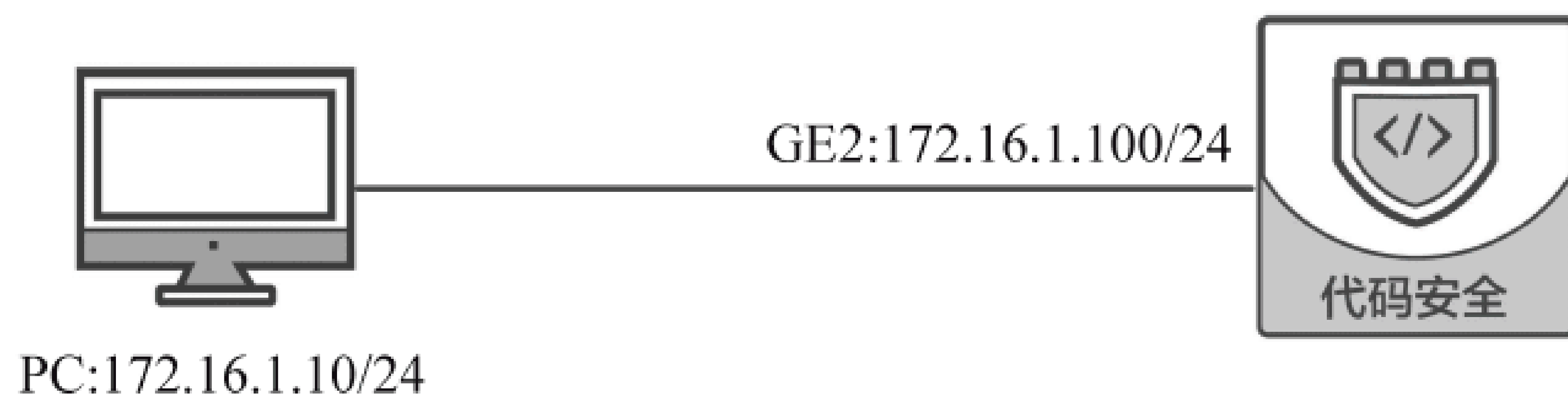


图 2-110　反射型 XSS 缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，则输入 123456。

(2) 打开“C:\code”目录下的 code.php 文件。

(3) 该程序的功能是 Web 服务器以 GET 方式读取用户输入的文章分类名称，不经任何过滤直接显示在用户浏览器中，如图 2-111 所示。

(4) 打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4  <title>文章分类</title>
5  </head>
6  <form action="" method="get">
7  <input type="text" name="name_input">
8  <input type="submit">
9  </form>
10 <?php
11 $name = $_GET['name_input'];
12 echo '请输入分类名称<br>'.$name;
13 ?>
14 </body>
15 </html>
```

图 2-111　代码解析(2.2.4)

(5) 单击“继续前往 172.16.1.100(不安全)”按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“反射型 XSS”,“开发语言”选中 PHP 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式,代码卫士平台只能上传“*.zip”文件),其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,单击“缺陷审计”按钮。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险为高,单击左侧框中的“+”按钮,展开目录,如图 2-112 所示。



图 2-112　展开界面(2.2.4)

（11）选择左侧的“code.php(12)”命令，可以快速定位有问题的代码。

（12）选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。信息表明用户所输入的数据都是不可信的，不对用户数据进行检验而直接显示在用户浏览器界面中会受到反射型 XSS 攻击，如图 2-113 所示。



图 2-113　详细信息(2.2.4)

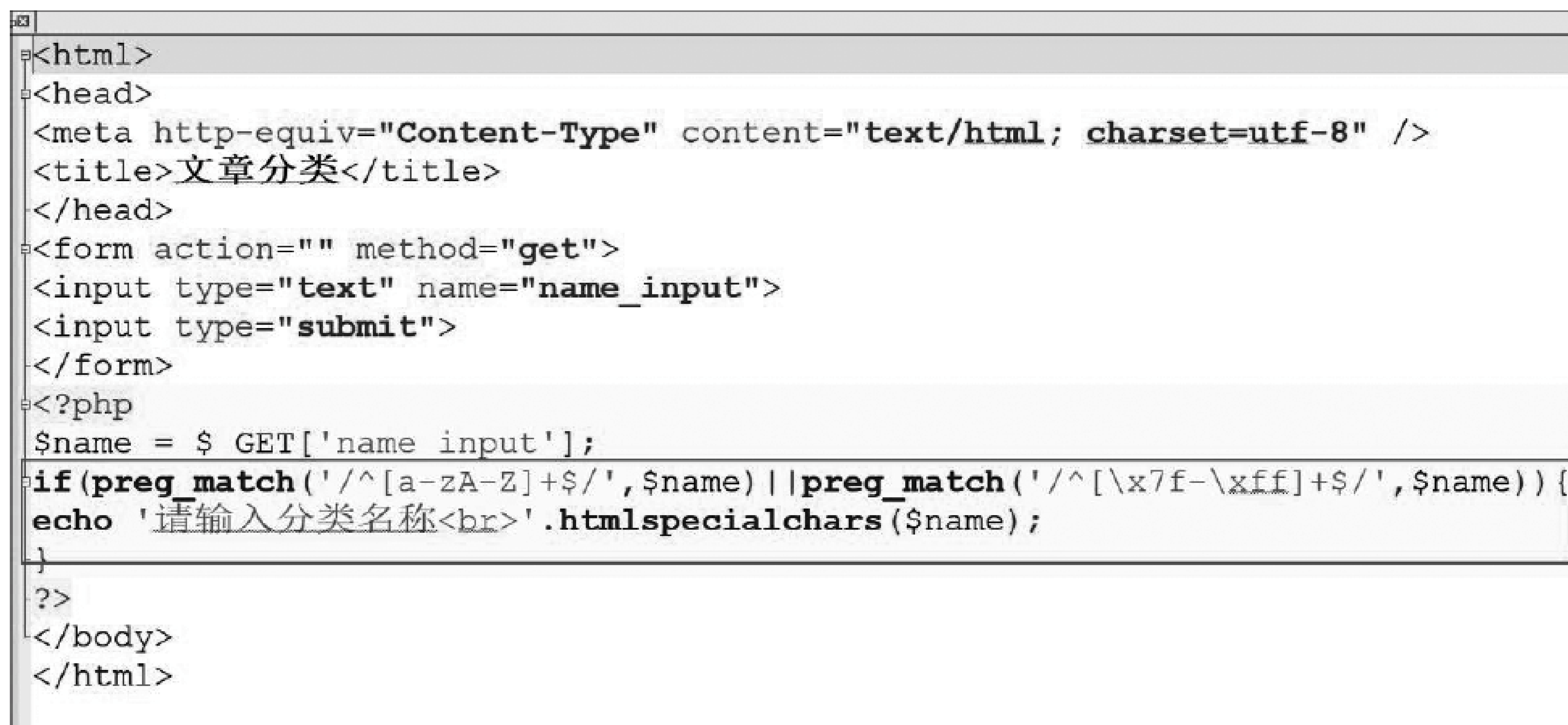
（13）选择“修复建议”命令，查看代码卫士给的修复建议。代码卫士建议通过对用户的输入数据进行验证、对要显示在浏览器界面上的数据进行 HTML 编码以及为 Cookie 设置 HTTPOnly 属性。

【实验预期】

修改后的代码再次被检测后等级降低。

【实验结果】

（1）最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，采用正则表达式只允许用户输入英文或汉字，否则输入的内容不显示在界面上，输出时对数据进行 HTML 编码，相关代码修改情况如图 2-114 所示，保存修改，关闭文件。



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>文章分类</title>
</head>
<form action="" method="get">
<input type="text" name="name_input">
<input type="submit">
</form>
<?php
$name = $_GET['name_input'];
if(preg_match('/^[a-zA-Z]+$/',$name)||preg_match('/^[\x7f-\xff]+$/',$name)){
echo '请输入分类名称<br>'.htmlspecialchars($name);
}
?>
</body>
</html>
```

图 2-114　代码修改(2.2.4)

(2) 右击 code.php,在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”,单击“立即压缩”按钮。

(4) 打开 Chrome 浏览器,在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“反射型 XSS 修复检测”,“开发语言”选中 PHP 单选钮,“源码来源”设置为“本地”,“上传文件”设置为“C:\code”目录下的“code-fix.zip”文件,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“等级分布”的图形示意为“中”,如图 2-115 所示。



图 2-115　检测完成(2.2.4)

(7) 可以看到风险等级已经降为“中”,如图 2-116 所示。



图 2-116　风险等级降低

【实验思考】

(1) 如何让用户只能输入数字、英文和汉字三种?

(2) 如何为 Cookie 设置 HTTPOnly 属性?

2.2.5　重定向缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测用户登录代码模块是否含有重定向缺陷,并对发现的重定向缺陷进行针对性的修复,确保所编写的代码中不含有重定向缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个用户登录代码模块,登录成功后会进行界面跳转。需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:重定向。

【实验原理】

重定向是指用户可以向 Web 站点提交指向其他站点的 URL,服务器会去执行,并将界面返回给用户。攻击者可能会使用 Web 服务器攻击其他站点,或者将 URL 换成一个钓鱼网站,安全意识薄弱的用户只看前面的网址,误以为是正常的站点,实则访问的是钓鱼网站,有可能遭受重大损失。

代码安全保障系统能够检测出代码中的缺陷,并给出详细描述以及修复建议,方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-117 所示。

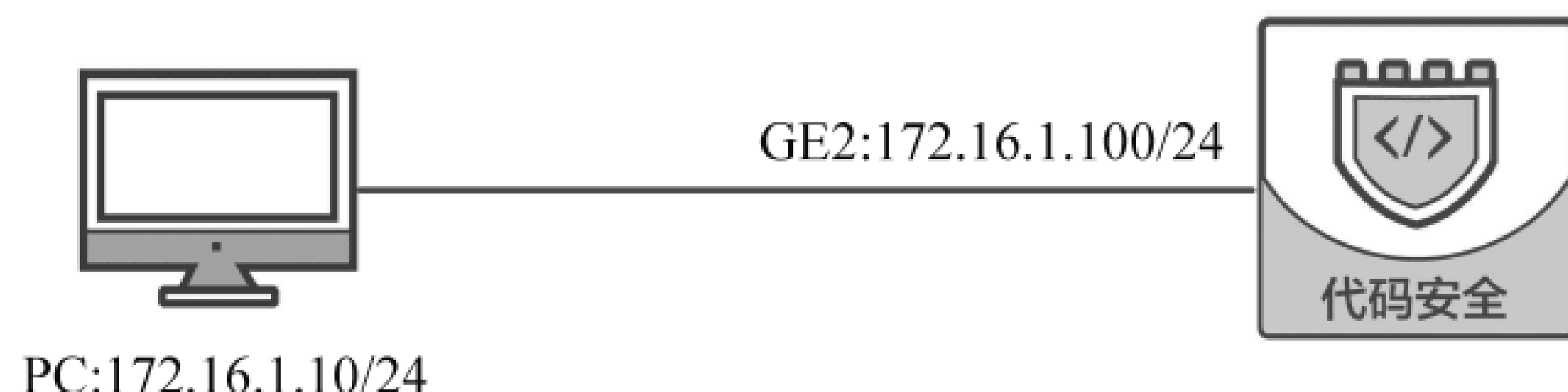


图 2-117　重定向缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测,检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,则输入 123456。

(2) 打开"C:\code"目录下的 code.php 文件。

(3) 该代码的功能是使用 GET 方法接受来自用户输入的 URL,然后将界面返回给用户,如图 2-118 所示。

```
1  <?php
2      session_start();
3      if(isset($_SESSION['login_ok']))
4      {
5      $url = $_GET["url"];
6      header("Location: " . $url);
7      }else{
8          echo "请先登录! ";
9          header("Location: " . login.php);
10     }
11     ?>
```

图 2-118　代码解析(2.2.5)

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(5) 单击"继续前往 172.16.1.100(不安全)"按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(7) 进入代码卫士后,选择"快速检测"→"+发起快速检测"命令。

(8) 在配置信息界面中,"任务名称"输入"重定向","开发语言"选中 PHP 单选钮,单击"上传文件"右侧的"浏览"按钮,选择"C:\code"目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式,代码卫士平台只能上传"*.zip"文件),其他保持默认配置,单击"发起检测"按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看"检测状态""缺陷总数"和"等级分布"等信息。当检测状态为"检测完成"时,单击"缺陷审计"按钮。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险等级为高。单击左

侧的“＋”，展开目录，如图 2-119 所示。



图 2-119　展开界面(2.2.5)

(11) 选择左侧的“code.php(6)”命令，可以快速定位有问题的代码。

(12) 选择“详细信息”命令，查看问题代码中该条缺陷的具体信息。信息表明未经验证的用户输入被当成重定向的 URL 或 URL 的一部分时，将导致重定向攻击，如图 2-120 所示。



图 2-120　详细信息(2.2.5)

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。建议表明防止重定向漏洞的方法是创建一份合法的 URL 列表，用户只能从中进行选择 URL 来进行重定向操作，或者创建一份合法的域名列表。

【实验预期】

修改后的代码再次被检测后，缺陷消除。

【实验结果】

(1) 最小化浏览器窗口,打开“C:\code”目录下的 code.php 文件,根据修复建议对代码进行修改,创建一份域名列表,使用户只能输入该列表中的 URL,相关代码修改情况如图 2-121 所示,保存修改,关闭文件。

```
1  <?php
2      session_start();
3      if(isset($_SESSION['login_ok'])){
4
5      $legal_urls = array("http://a.example.com", "http://b.example.com", "http://c.example.com");
6      $if(in_array($_GET['url'], $legal_urls, true)){
7              header("Location: " . $_GET['url']);
8      }else{
9          echo "请先登录!";
10         header("Location: " . login.php);
11     }
12     ?>
```

图 2-121　代码修改(2.2.5)

(2) 右击 code.php,在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”,选择“立即压缩”命令。

(4) 打开 Chrome 浏览器,在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“重定向修复检测”,“开发语言”选中 PHP 单选钮,“源码来源”设置为“本地”,“上传文件”设置为“C:\code”目录下的“code-fix.zip”文件,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“等级分布”的图形示意为空,缺陷总数为 0,如图 2-122 所示。

代码卫士 codesafe　首页　快速检测　项目管理　报告管理　统计分析　系统管理　当前用户:admin　资源下载

缺陷检测　合规检测　溯源检测　+发起快速检测

任务列表　搜索

任务名称	开发语言	检测开始时间	检测结束时间	检测状态	缺陷总数	等级分布	创建者	删除选中项
重定向修复检测	PHP	2018-03-18 19:13:37	2018-03-18 19:14:02	检测完成	0		admin	
重定向	PHP	2018-03-18 19:07:24	2018-03-18 19:07:49	检测完成	2		admin	

图 2-122　检测完成(2.2.5)

【实验思考】

(1) 搭建 Web 站点,尝试复原漏洞,使用户通过该站点访问百度界面。

(2) 创建一份域名列表,使用户只能访问域名列表中的网站。

2.2.6　路径遍历缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测日志文件管理代码模块是否有路径遍历缺陷，并对发现的路径遍历缺陷进行针对性的修复，确保所编写的代码没有路径遍历缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个日志文件管理代码模块，定期对日志文件进行清理。需要使用代码安全保障系统对编写的代码进行缺陷检测，并通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：路径遍历。

【实验原理】

Web 应用程序通常会有对系统文件操作的功能，常常需要用到提交的参数来指明文件名，例如 http://www.example.com/getfile=image.jpg，当服务器处理传送过来 image.jpg 文件名后，Web 应用程序即会自动添加完整路径，如“D://site/images/image.jpg”，将读取的内容返回给访问者。但是由于文件名可以在客户端随意更改，攻击者利用服务器的特性，比如通过特殊符号“～/”“../”等进行目录跳转回溯，从而可以越权访问或者覆盖敏感数据，如网站的配置文件、系统的核心文件，这样的缺陷被命名为路径遍历缺陷。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-123 所示。

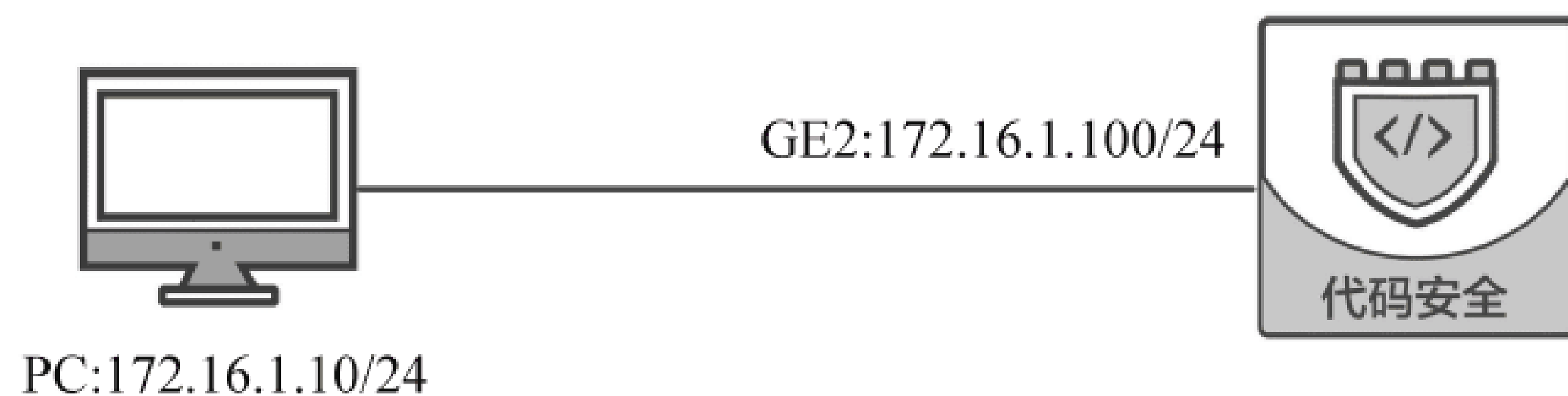


图 2-123　路径遍历缺陷检测实验拓扑图

【实验思路】

（1）发起 PHP 代码缺陷检测。

（2）根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code”目录下的 code.php 文件。

(3) 该代码的功能是通过读取用户输入的文件名，之后判断文件是否存在，存在则将其删除。从代码中可以看到程序并未对用户输入的数据进行任何过滤或限制，极易引起路径遍历漏洞的发生，如图 2-124 所示。

```
code.php
1  <?php
2
3  $filename= $_GET['filename'];
4
5  if(is_file($filename)){
6      unlink($filename);
7  }
8  ?>
```

图 2-124　代码解析(2.2.6)

(4) 打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100(不安全)”按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“路径遍历”，“开发语言”选中 PHP 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式，代码卫士平台只能上传“*.zip”文件)，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时，单击“缺陷审计”按钮。

(10) 可以看到代码卫士检测出了相关缺陷，并判断该缺陷的风险等级为“低”。单击左侧框中的“+”，展开目录，如图 2-125 所示。

(11) 选择左侧的“code.php(6)”命令，可以快速定位有问题的代码。

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。详细信息中解释了何时会发生路径遍历漏洞以及攻击者会如何利用此漏洞进行攻击，如图 2-126



图 2-125　展开界面(2.2.6)

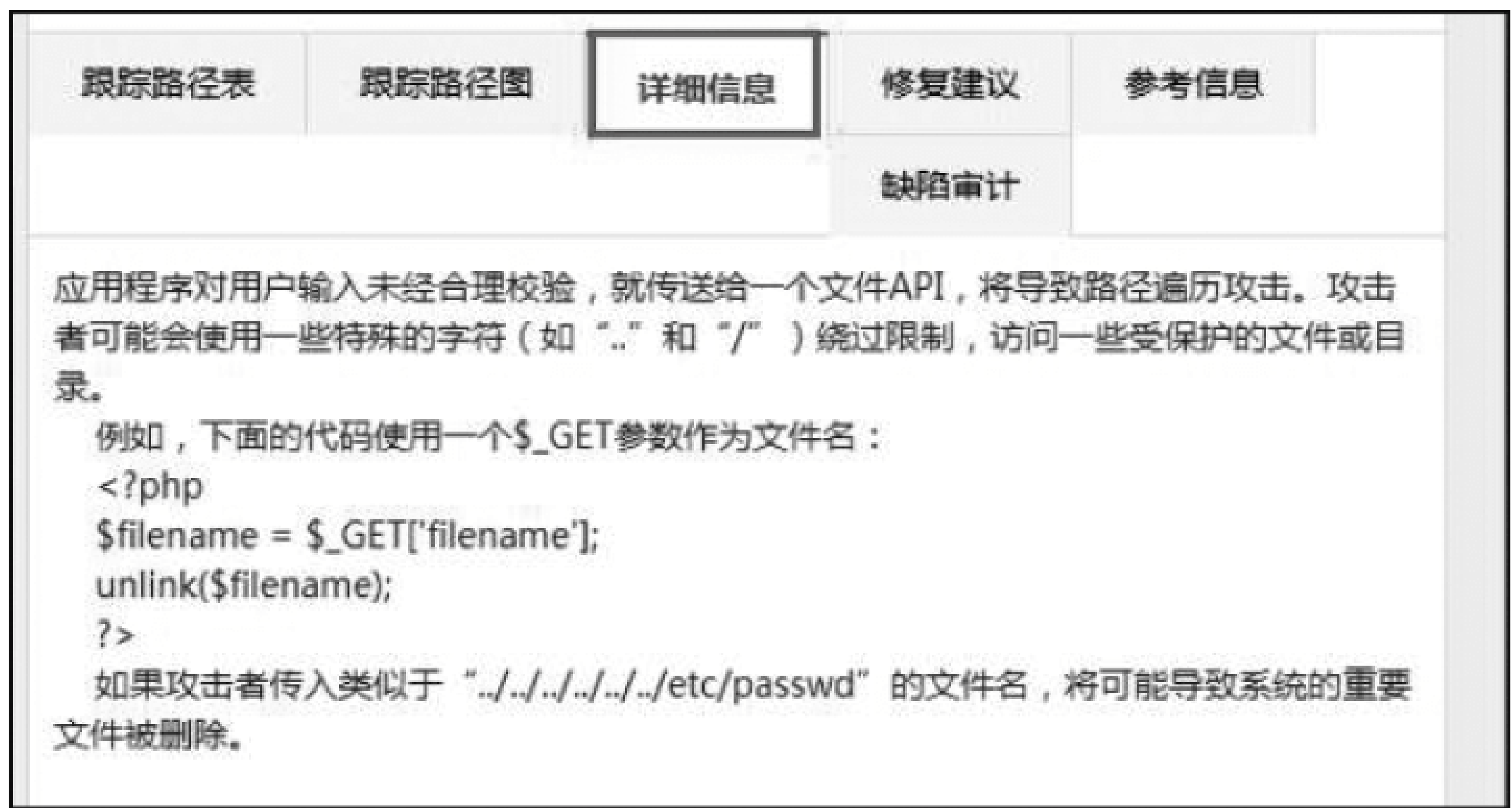


图 2-126　详细信息(2.2.6)

所示。

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。建议表明防止路径遍历漏洞的最佳方法是创建一份合法资源名的列表或者创建一份合法的字符列表，只允许用户删除列表中指定的文件或只允许输入指定的字符。

【实验预期】

修改后的代码再次被检测后缺陷消除。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php。根据修复建议，创建了一份文件列表，只允许用户删除文件列表中的文件，相关代码修改情况如图 2-127 所示，保存修改，关闭文件。

```php
<?php
    $fileindex = intval($_GET["fileindex"]);

    $files = array("a.txt", "b.txt", "c.txt");

    if(is_file($files[$fileindex])){

    unlink("/tmp/". $files[$fileindex]);

    }
    ?>
```

图 2-127　代码修改(2.2.6)

(2) 右击 code.php,在弹出的快捷菜单中选择"添加到压缩文件(A)…"命令。

(3) "压缩到"中的文件名称输入"code-fix.zip",选择"立即压缩"命令。

(4) 打开 Chrome 浏览器,在之前的界面上选择"快速检测"→"+发起快速检测"命令。

(5) 在配置信息界面中,"任务名称"输入"路径遍历修复检测","开发语言"选中 PHP 单选钮,"源码来源"设置为"本地","上传文件"设置为"C:\code"目录下的"code-fix.zip"文件,其他保持默认配置,单击"发起检测"按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,"等级分布"的图形示意为空,缺陷总数为 0,如图 2-128 所示。

代码卫士 codesafe　首页　快速检测　项目管理　报告管理　统计分析　系统管理　当前用户:admin　资源下载

缺陷检测　合规检测　溯源检测　+ 发起快速检测

任务列表　搜索

	任务名称	开发语言	检测开始时间	检测结束时间	检测状态	缺陷总数	等级分布	创建者	删除选中项
	路径遍历修复检测	PHP	2018-03-18 17:33:53	2018-03-18 17:34:17	检测完成	0		admin	
	路径遍历	PHP	2018-03-18 17:27:13	2018-03-18 17:27:39	检测完成	1		admin	

图 2-128　检测完成(2.2.6)

【实验思考】

(1) 搭建 Web 环境,将未修改的代码文件放在搭建好的网站中,尝试复现此漏洞。

(2) 创建一份合法的字符列表,只允许用户输入字符列表中的字符。

2.2.7 动态解析代码缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测计算器代码模块中是否含有动态解析缺陷，并对发现的动态解析缺陷进行针对性的修复，确保所编写的代码模块中不含有动态解析缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个计算器代码模块，对用户输入的表达式进行求值。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：动态解析代码。

【实验原理】

eval()函数的原型为 mixed eval(string $code)，该函数可以将字符串 code 作为 php 代码执行。代码执行的作用域是 eval()处的作用域，因此，eval()中的任何的变量定义、修改，都会在函数结束后被保留。php 手册表明函数 eval()的语言结构是非常危险的，因为它允许执行任意的 php 代码。若除了使用此结构无其他方法，请多加注意，不要允许传入任何用户提供的、未经完整验证过的数据。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-129 所示。

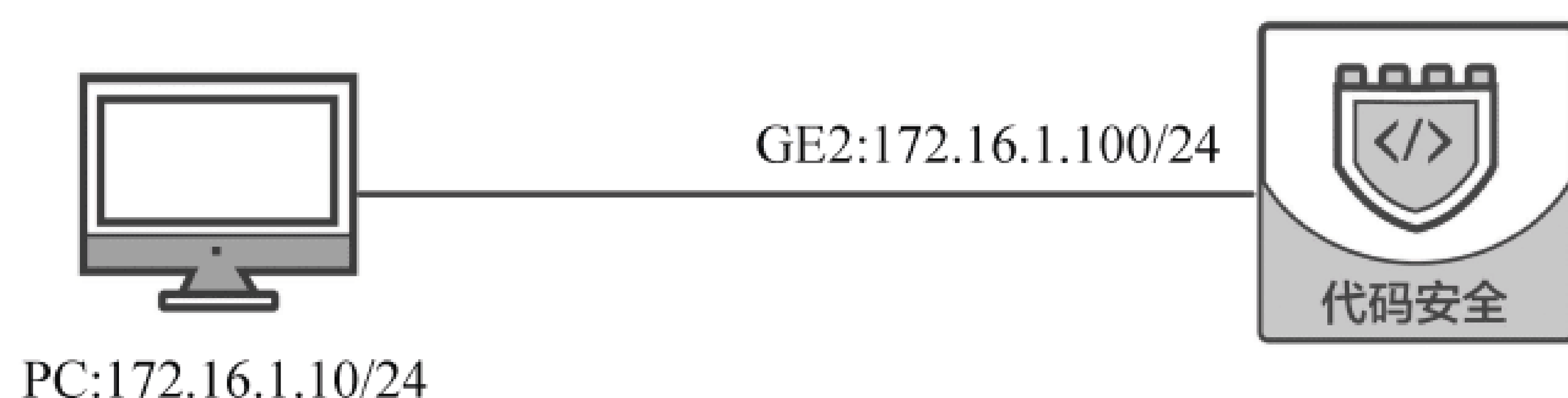


图 2-129 动态解析代码缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开“C:\code”目录下的 code.php 文件。

(3) 该代码的功能是通过 GET 方式获取用户输入的两个操作数 op1 和 op2 以及一个操作符,将获取到的操作符直接使用函数“eval()”赋值给变量“$operator”,然后计算表达式的值。函数“eval()”可以将所给参数解析成 php 命令并执行,如图 2-130 所示。

```
1 <?php
2       $op1 = $_GET['op1'];
3       $op2 = $_GET['op2'];
4       $operator = $_GET['operator'];
5       eval("\$operator = $operator;");
6       ?>
```

图 2-130 代码解析(2.2.7)

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100(不安全)”按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“动态解析”,“开发语言”选中 PHP 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的 code.zip 文件,其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险等级为高。单击左侧框中的“+”按钮,展开目录,如图 2-131 所示。

(11) 选择左侧的“code.php(5)”命令,可以快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。信息表明 PHP 允许在运行时动态解析执行源代码,若未对用户输入的参数进行过滤,当这一功能被攻击者利用时会造成代码注入攻击,如图 2-132 所示。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。建议表明任何时候都不要直接解析执行未经验证的用户输入,而应该创建一份白名单,用户只能从中选择需要进行的动作或者需要使用的数据。



图 2-131　展开界面(2.2.7)

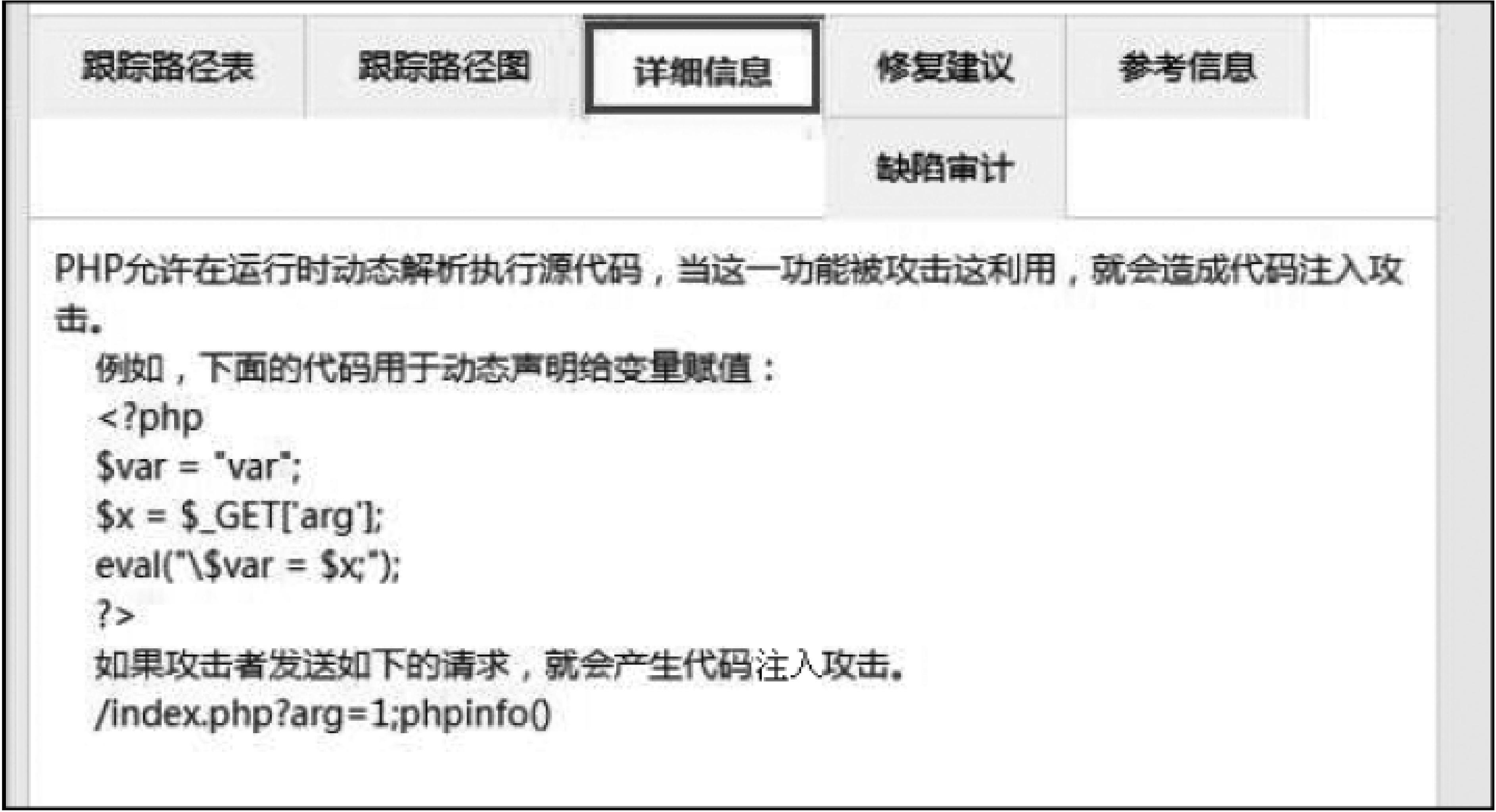


图 2-132　详细信息(2.2.7)

【实验预期】

修改后的代码再次被检测后，缺陷消除。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，创建一份白名单，名单只允许用户使用“+”“-”“*”“/”这 4 种运算符，(相关代码修改情况如图 2-133 所示)，从而避免程序执行其他命令给系统造成损失，保存修改，关闭文件。

(2) 右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，选择“立即压缩”命令。

(4) 打开 Chrome 浏览器，在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“动态解析修复检测”，“开发语言”选中 PHP 单选钮，“源码来源”设置为“本地”，“上传文件”设置为“C:\code”目录下的“code-fix.zip”文件，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“等级分布”的图像

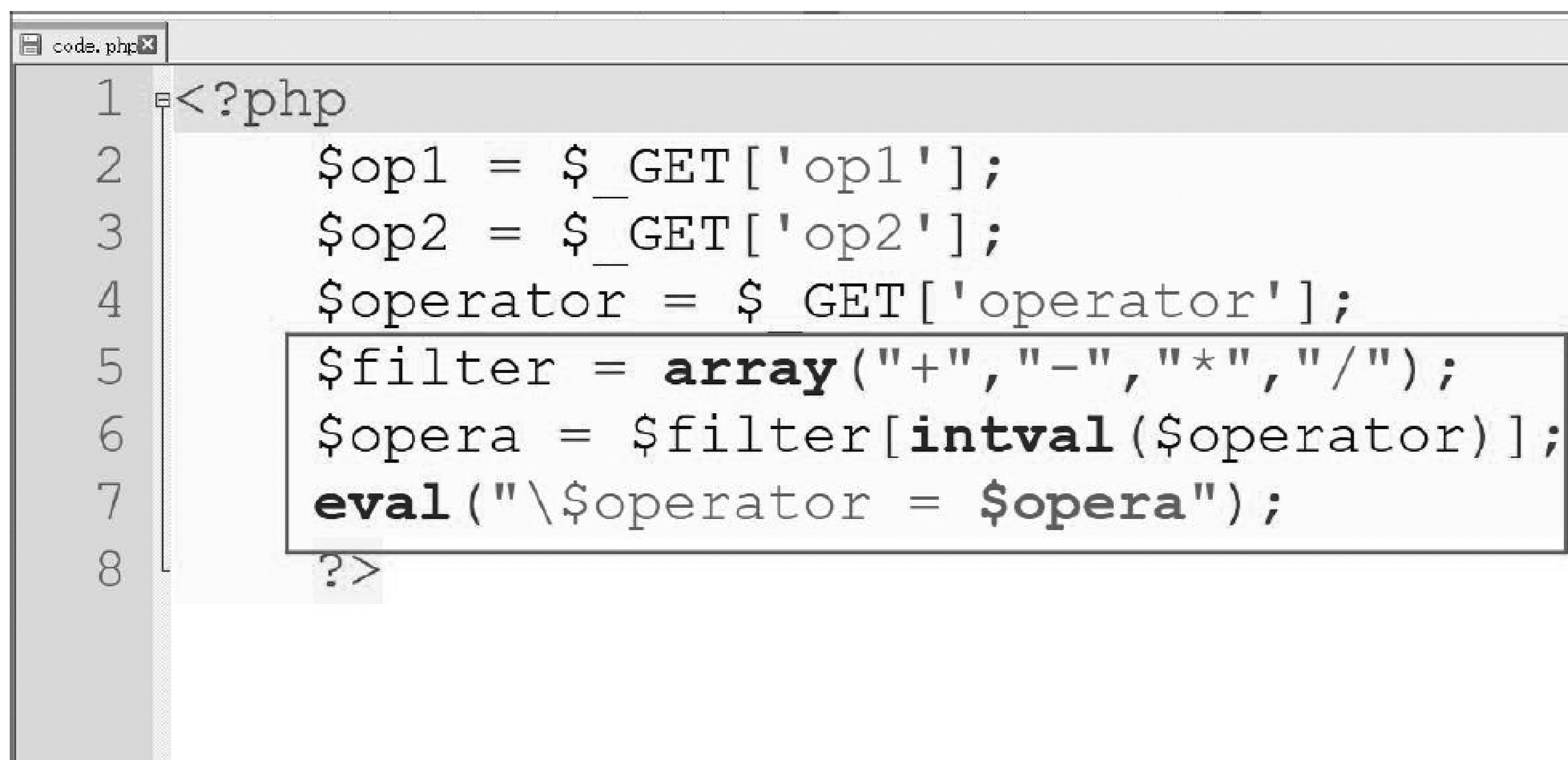


图 2-133　代码修改(2.2.7)

示意为空,缺陷总数为 0,如图 2-134 所示。



图 2-134　检测完成(2.2.7)

【实验思考】

(1) 搭建 Web 网站环境,将修改之前的代码放入网站目录中,尝试复原漏洞,使界面显示 phpinfo 的信息。

(2) 尝试使用正则表达式使用户的操作符只能输入"+""-""*""/"这 4 个字符。

2.2.8　不安全的哈希算法缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测用户会话管理模块是否有不安全的哈希算法,并对发现的不安全的哈希算法进行针对性的修复,确保所编写的代码中不存在不安全的哈希算法。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个用户会话管理代码模块，使用哈希算法来生成用户的会话 ID。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：不安全的哈希算法。

【实验原理】

Hash 常翻译为“散列”，也可以音译为“哈希”，就是把任意长度的输入通过散列算法变换成固定长度的输出，用以提供消息的完整性保护。

一个安全的哈希算法应当具有足够长的位数，避免能在可接受时间内寻找到有意义的哈希碰撞。

常见的哈希算法有 MD5、SHA-1、SHA-224、SHA-256 等。其中，MD5 和 SHA-1 先后在 2004 年和 2005 年被破解，因此这两个哈希算法被业界认为是不安全的。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-135 所示。

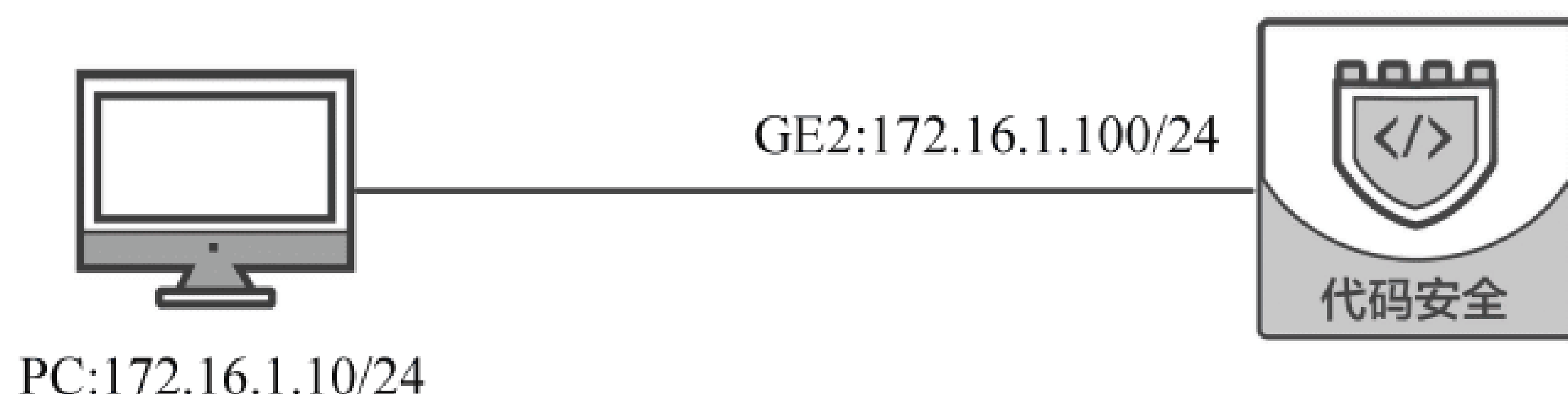


图 2-135　不安全的哈希算法缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code”目录下的 code.php 文件。

(3) 该代码将用户输入的密码参数“$password”经过 MD5 算法后与数据库中的值进行比较，可见数据库中密码是以 MD5 散列值存在的，容易根据散列值构造出碰撞的密码值，或者通过彩虹表进行破解，如图 2-136 所示。

```php
<?php
    $user = $_POST['username'];
    $password = $_POST['password'];
    if($user == $db_user && md5($password) === $db_pass){
        //auth success
    }
    ?>
```

图 2-136　代码解析(2.2.8)

(4) 打开 Chrome 浏览器，输入代码卫士的 IP 地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100(不安全)”按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不安全的哈希算法”，“开发语言”选中 PHP 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时，单击“缺陷审计”按钮。

(10) 可以看到代码卫士检测出了相关缺陷，并判断该缺陷的风险等级为低。单击左侧框中的“+”按钮，展开目录，如图 2-137 所示。



图 2-137　展开界面(2.2.8)

(11) 选择左侧的“code.php(4)”命令，可以快速定位有问题的代码。

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。信息表明 MD5 和 SHA-1 都属于不安全的哈希算法，不应该再用来保证数据的完整性，如图 2-138 所示。



图 2-138　详细信息(2.2.8)

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。建议表明在安全性要求较高的系统中，应采用散列值>=224 比特的 SHA 系列算法(如 SHA-22(4)、SHA-256 等)。

【实验预期】

修改后的代码再次被检测后，缺陷消除。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，使用 SHA-256 哈希算法代替 MD5(相关代码改为 hash(sha256, $password))，保存修改并关闭文件，如图 2-139 所示。

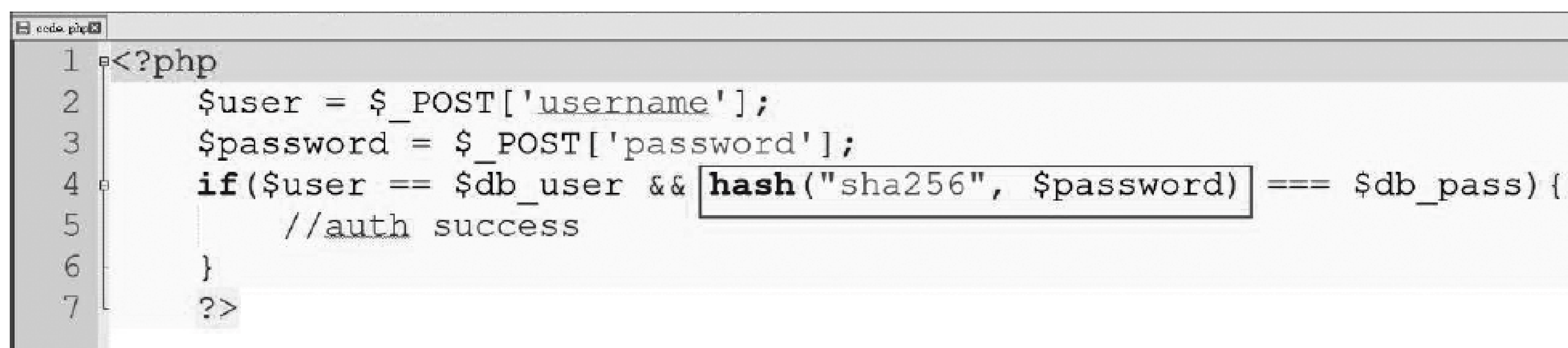


图 2-139　代码修改(2.2.8)

(2) 右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开 Chrome 浏览器，在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“不安全的哈希算法修复检测”，“开发语言”选中 PHP 单选钮，“源码来源”设置为“本地”，“上传文件”设置为“C:\code”目录下的

"code-fix.zip"文件,其他保持默认配置,单击"发起检测"按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,"等级分布"的图形示意为空,缺陷总数为0,如图2-140所示。



图 2-140 检测完成(2.2.8)

【实验思考】

(1) 将MD5换成SHA-1再次检测,查看检测结果。

(2) 将MD5换成SHA-224再次检测,查看检测结果。

2.2.9 XPath 注入缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测工资管理代码模块是否含有XPath注入缺陷,并对发现的XPath注入缺陷进行针对性的修复,确保代码模块中不存在XPath注入缺陷。

【知识点】

PHP代码缺陷检测。

【场景描述】

A公司研发部工程师小王编写了一个工资管理代码模块,工资数据使用XML存储,使用Xpath来查询员工数据。需要使用代码安全保障系统对编写的代码进行缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测:XPath注入。

【实验原理】

XPath注入攻击本质上和SQL注入类似,都是输入一些恶意的查询代码字符串来对网站进行攻击。

XPath 注入攻击,是指利用 XPath 解析器的松散输入和容错特性,能够在 URL、表单或其他信息上附带恶意的 XPath 查询代码,以获得权限信息的访问权并更改这些信息。XPath 注入攻击是针对 Web 服务应用新的攻击方法,它允许攻击者在事先不知道 XPath 查询相关信息的情况下,通过 XPath 查询得到 XML 文档的完整内容。

代码安全保障系统能够检测出代码中的缺陷,并给出详细描述以及修复建议,方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-141 所示。

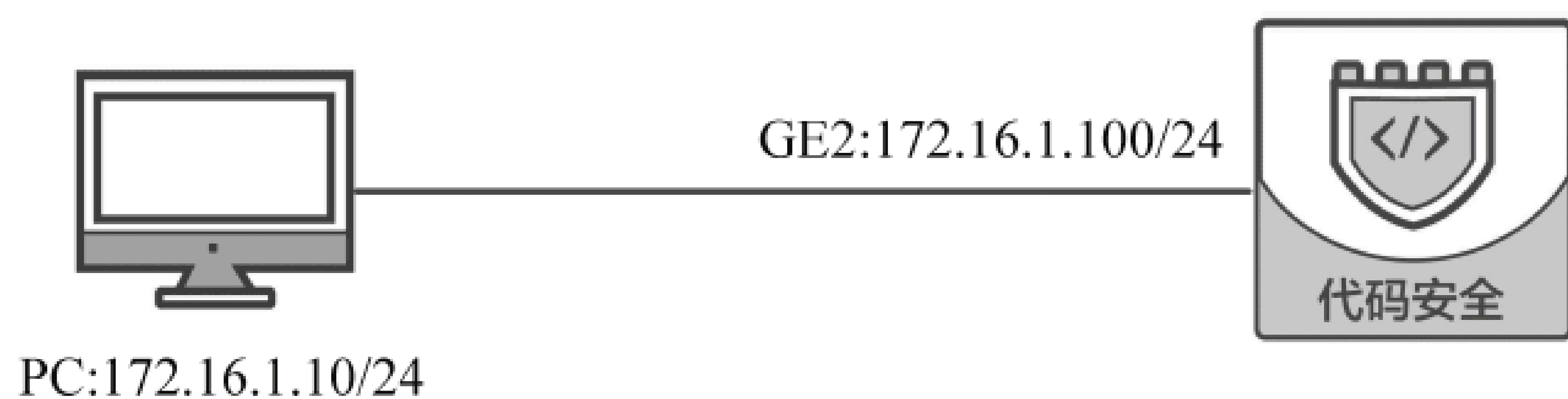


图 2-141　XPath 注入缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测,检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开"C:\code"目录下的 code.php 文件。

(3) 该代码的功能是使用 GET 方法获取用户输入的用户名和密码,然后使用 XPATH 语法与 xml 文件中存储的用户名、密码比对,比对成功则登录成功,如图 2-142 所示。

```
1  <?php
2  $doc = new DOMDocument();
3  $doc->load('employees.xml');
4  $xpath = new DOMXPath($doc);
5  $results = $xpath->query("//Employee[UserName='" . $_GET["Username"] . "'][Password='" . $_GET["Password"] . "']");
6  if($results !== false && $results->length > 0){
7          //Login success
8      }
9      ?>
```

图 2-142　代码解析(2.2.9)

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100(不安全)”菜单命令。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“XPath 注入”,“开发语言”选中 PHP 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code.zip”文件,其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,单击“缺陷审计”按钮。

(10) 可以看到代码卫士检测出了相关缺陷,并判断该缺陷的风险等级为高。单击左侧框中的“+”按钮,展开目录,如图 2-143 所示。



图 2-143　展开界面(2.2.9)

(11) 选择左侧的“code.php(5)”命令,可以快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。信息表明使用不可信数据源的数据来构造并执行 XPath 查询,就有可能会发生 XPath 注入攻击,攻击者可以利用此方式获取未授权的数据,或者篡改这些数据,如图 2-144 所示。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。建议解释了发生 XPath 注入的原因并建议通过对用户输入的数据进行过滤,去掉那些可能用作 XPath 命令的字符。

【实验预期】

修改后的代码再次被检测后,缺陷消除。



图 2-144　详细信息(2.2.9)

【实验结果】

修改后的代码再次被检测后缺陷消除

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，只允许用户输入数字，保存修改，关闭文件，如图 2-145 所示。

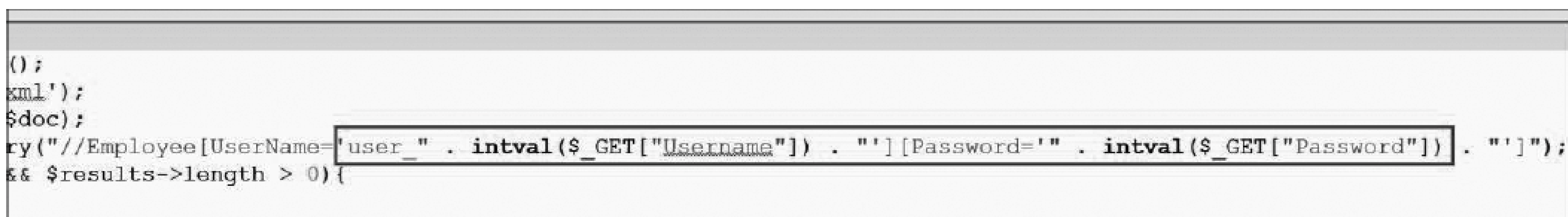


图 2-145　代码修改(2.2.9)

(2) 右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)...”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，选择“立即压缩”命令。

(4) 打开 Chrome 浏览器，在之前界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“XPath 注入修复检测”，“开发语言”选中 PHP 单选钮，“源码来源”设置为“本地”，“上传文件”设置为“C:\code”目录下的“code-fix.zip”文件，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“等级分布”的图形示意为空，缺陷总数为 0，如图 2-146 所示。



图 2-146　检测完成(2.2.9)

【实验思考】

(1) 搭建 Web 环境，书写一个存有账号密码的 xml 文件，尝试复现此漏洞。

(2) 修改代码，使用户可以输入字母和数字。

2.2.10　硬编码密码缺陷检测实验

【实验目的】

通过使用代码安全保障系统检测数据库连接管理代码模块是否含有硬编码密码缺陷，并对发现的硬编码密码缺陷进行针对性的修复，确保所编写的代码中不含有硬编码密码缺陷。

【知识点】

PHP 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个数据库连接管理代码模块，通过用户名和密码连接数据库并读取数据。需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：硬编码密码。

【实验原理】

硬编码密码是指在编写数据库连接模块中，开发人员直接将数据库用户名和密码写在源代码中。这是一种非常不好的编码习惯，使得其他可以接触到源代码的人员很容易对整个数据库进行控制。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-147 所示。

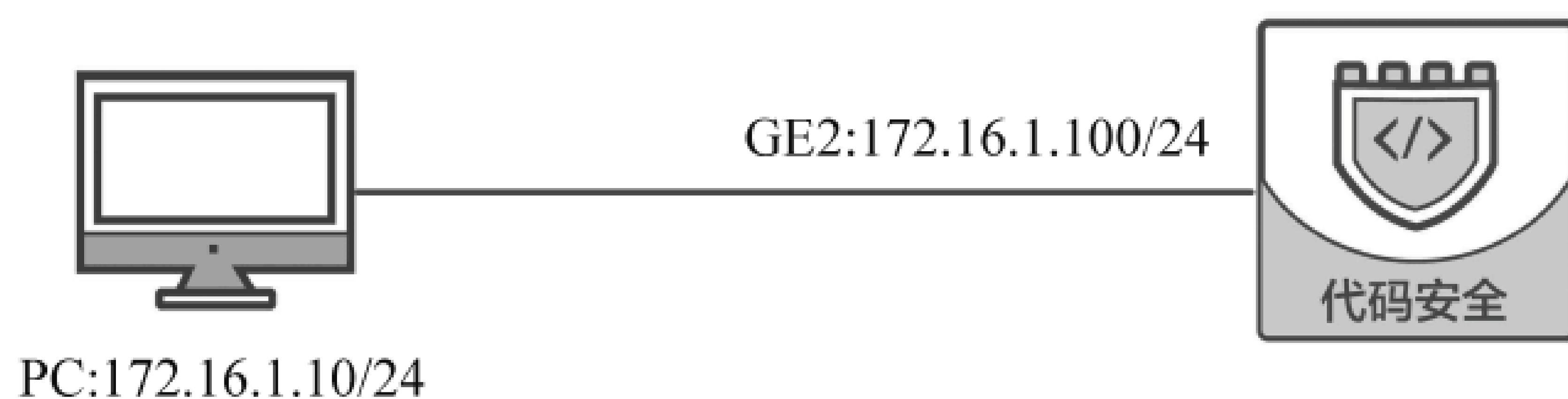


图 2-147　硬编码密码缺陷检测实验拓扑图

【实验思路】

(1) 发起 PHP 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 对修改后的 PHP 代码发起缺陷检测,检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开"C:\code"目录下的 code.php 文件,如图 2-148 所示。



图 2-148　打开 code.php 文件

(3) 该代码的功能是连接 MySQL 数据库。通过代码可以看到数据库用户名和密码都写在了程序源码中,容易被人窃取,如图 2-149 所示。

(4) 打开 Chrome 浏览器,输入代码卫士的 IP 地址"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(5) 单击"继续前往 172.16.1.100(不安全)"按钮。

(6) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(7) 进入代码卫士后,选择"快速检测"→"+发起快速检测"命令。

(8) 在配置信息界面中,"任务名称"输入"硬编码密码","开发语言"选中 PHP 单选钮,单击"上传文件"右侧的"浏览"按钮,选择"C:\code"目录下的 code.zip 文件(此文件是 code.php 文件的压缩包格式,代码卫士平台只能上传 *.zip 文件),其他保持默认配置,单

```php
<?php
$user = 'root';

$password = '123456';

$con=mysqli_connect("localhost",$user,$password,"RUNOOB");
if (mysqli_connect_errno($con))
{
    echo "连接 MySQL 失败: " . mysqli_connect_error();
}

// 执行查询
mysqli_query($con,"SELECT * FROM websites");
mysqli_query($con,"INSERT INTO websites (name, url, alexa, country)
VALUES ('百度','https://www.baidu.com/','4','CN')");

mysqli_close($con);
?>
```

图 2-149　代码解析

击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时，选择“缺陷审计”命令(图中最右框位置)，如图 2-150 所示。



图 2-150　开始检测

(10) 可以看到代码卫士检测出了相关缺陷，并判断该缺陷的风险等级为高。单击左侧框中的“+”按钮展开目录，如图 2-151 所示。

(11) 选择左侧的“code.php(4)”按钮，可以快速定位有问题的代码，如图 2-152 所示。

(12) 选择“详细信息”命令可以查看问题代码中该条缺陷的具体信息。详细信息中举例说明了硬编码密码对系统的威胁。任何对该代码具有访问权限的人都能获取这个密码。并且从代码维护的角度来看，若以后需要修改这个密码，则必须要修改源代码，这在生产环境中并不是一个好的选择，如图 2-153 所示。

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。对于重要的密码信息，应该采取人工输入或从其他安全的外部渠道来获取，如图 2-154 所示。



图 2-151　展开界面

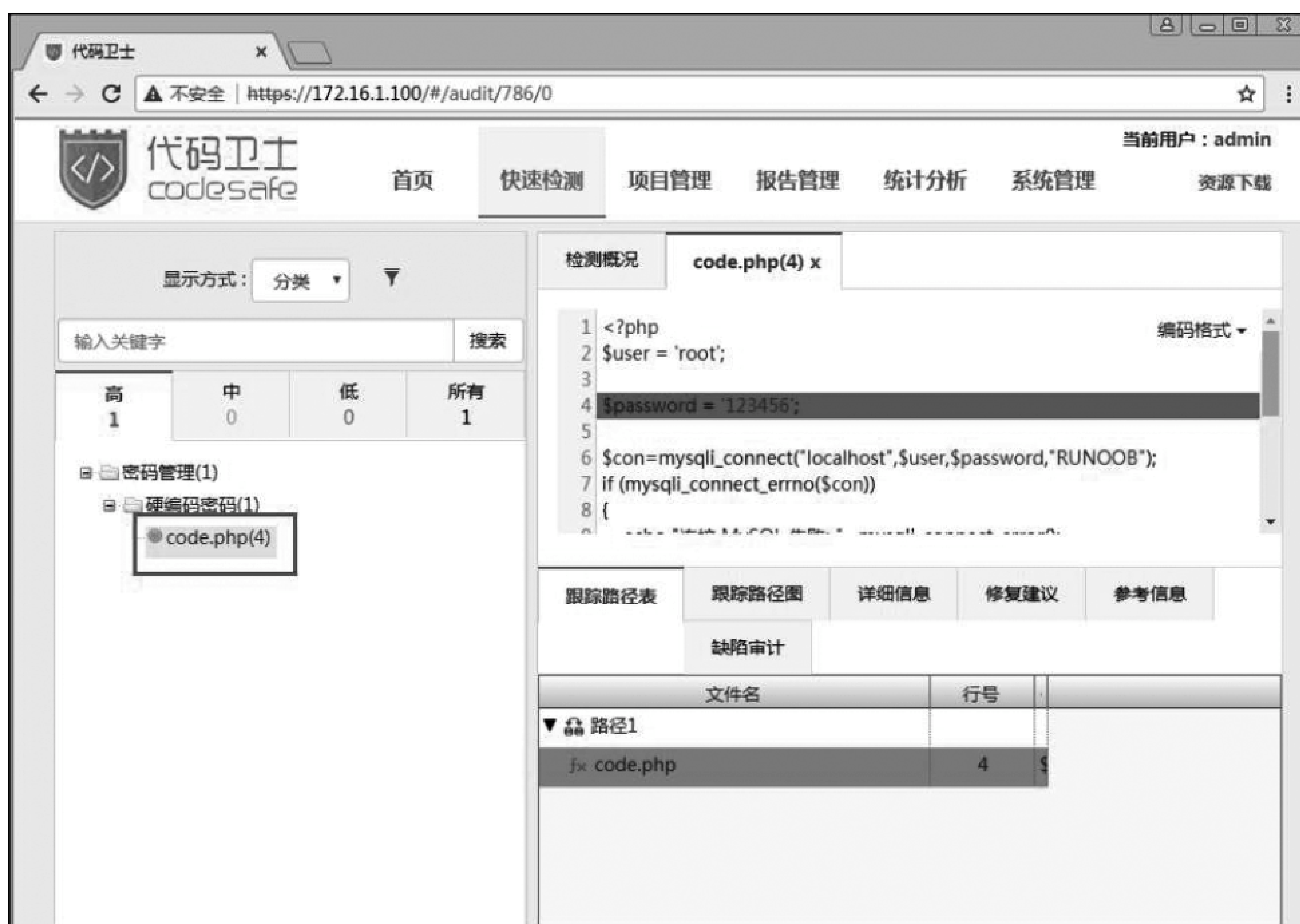


图 2-152　定位有问题的代码

【实验预期】

修改后的代码再次被检测后缺陷消除。



图 2-153　详细信息



图 2-154　修复建议

【实验结果】

（1）最小化浏览器窗口，在“C:\code”目录下新建文本文件 config.inc.txt，并写入“$user = 'root'; $password = '123456';”（在实际环境中，该文件应该保存到 Web 服务器目录之外，防止恶意用户直接通过浏览器读取到该文件。若有可能，可以对该文件加密并设置访问权限，从而避免代码被他人随意看到，本实验不再演示此操作），保存并关闭文件。打开“C:\code”目录下的 code.php 文件，根据修复建议，使用“require_once()”函数

引用 config.inc.txt 文件。保存修改，关闭文件，如图 2-155 和图 2-156 所示。



图 2-155　代码修改(2.2.10)



图 2-156　配置文件

(2) 右击 code.php 和 config.inc.txt，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令，如图 2-157 所示。

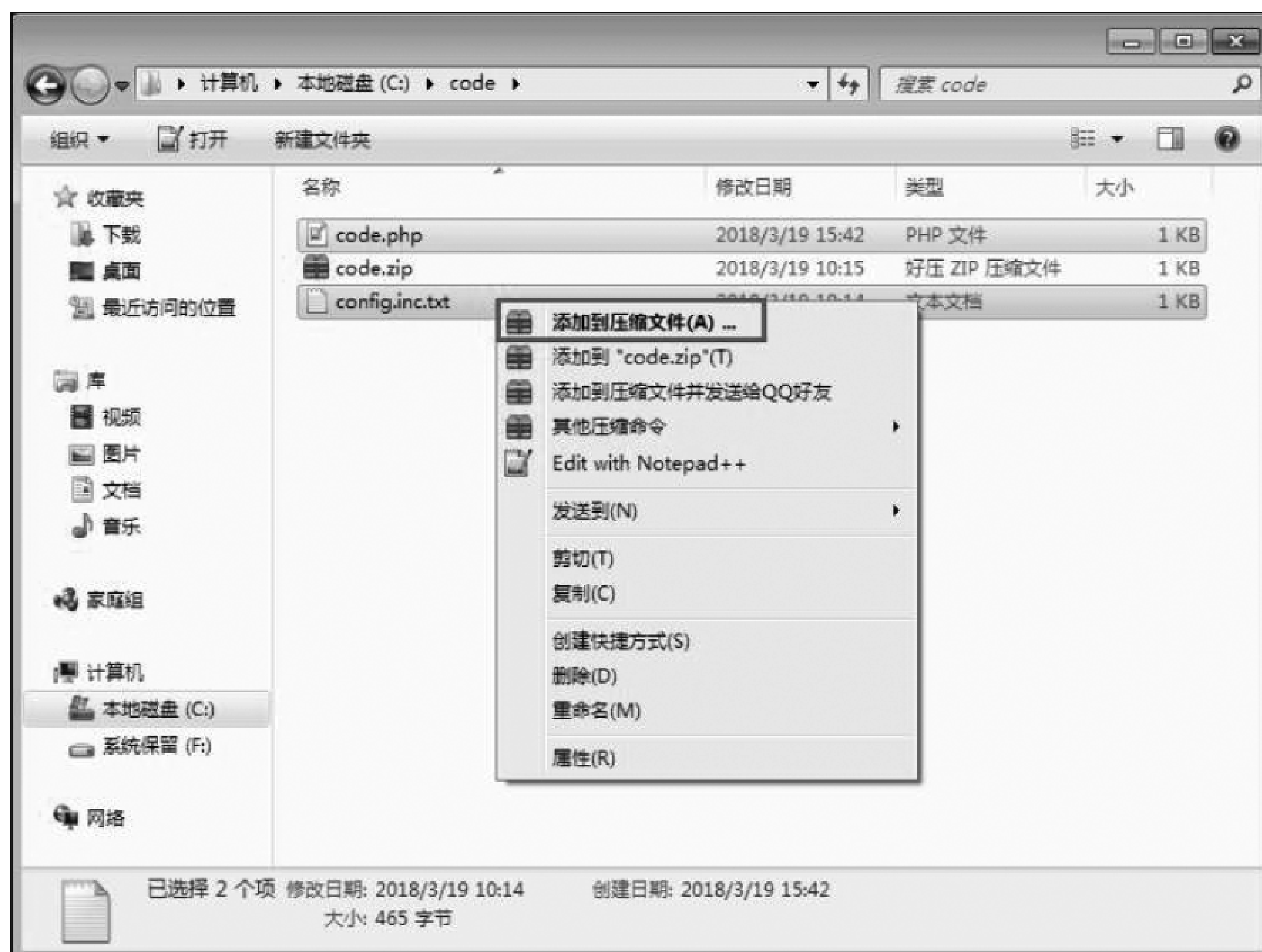


图 2-157　添加压缩文件

(3)“压缩到”中的文件名称输入 code-fix.zip,单击“立即压缩”按钮,如图 2-158 所示。



图 2-158　压缩成 code-fix.zip

(4)打开 Chrome 浏览器,在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5)在配置信息界面中,“任务名称”输入“硬编码密码修复检测”,“开发语言”选中 PHP 单选钮,“源码来源”设置为“本地”,“上传文件”选择“C:\code”目录下的“code-fix.zip”文件,其他保持默认配置,单击“发起检测”按钮。

(6)在任务列表界面等待检测完成,可以看到该项目检测完成后,“等级分布”的图形示意为空,缺陷总数为 0,如图 2-159 所示。

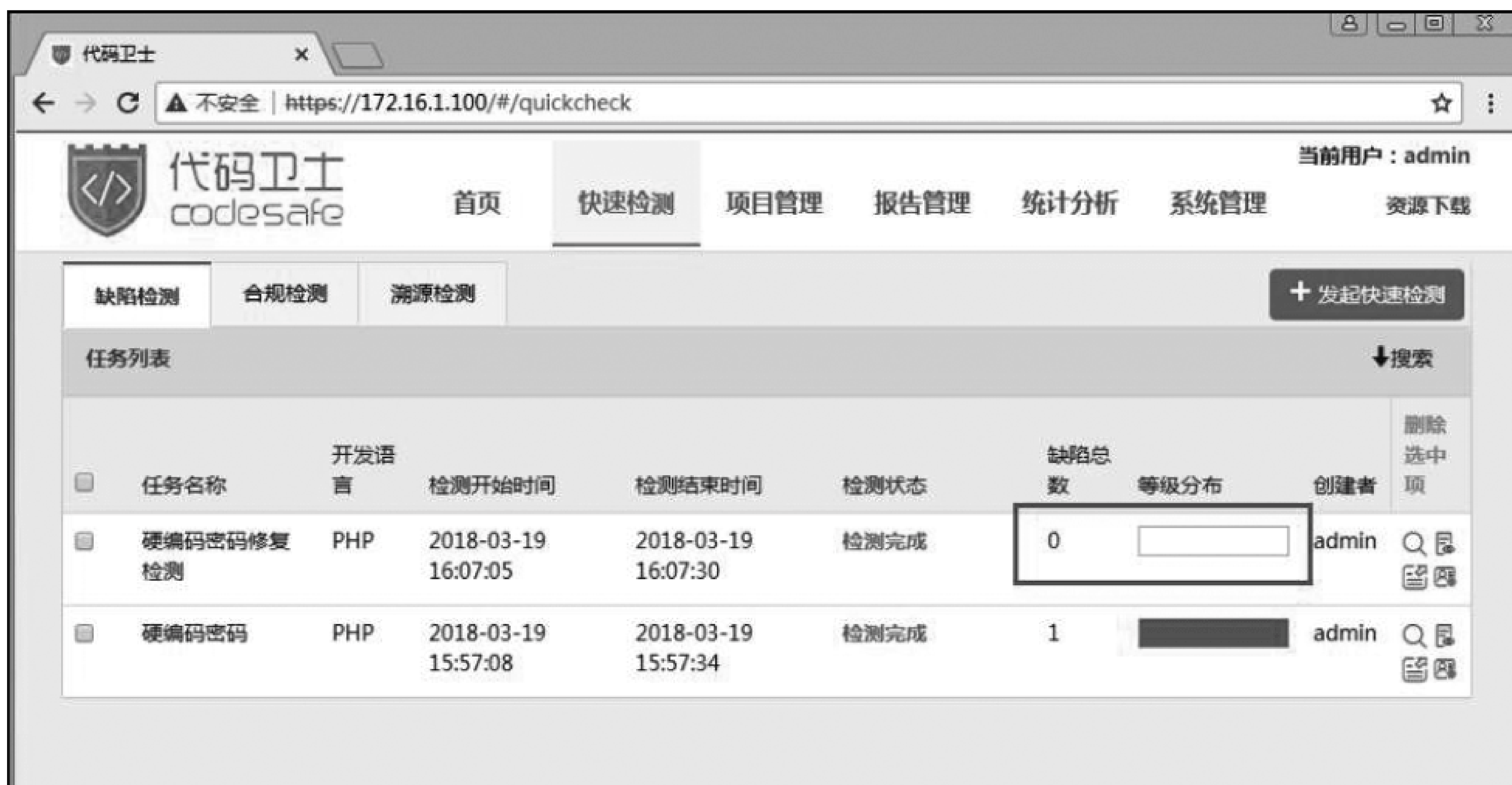


图 2-159　检测完成(2.2.10)

【实验思考】

(1)尝试对实验中配置文件设置访问权限,仅系统管理员可以访问。

(2)尝试对配置文件中的内容进行加密。

2.3　Java 缺陷检测

本节主要完成文件资源来释放缺陷检测实验。

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“资源未释放：文件”的缺陷，并对发现的“资源未释放：文件”的缺陷进行针对性的修复，确保所编写的代码中不存在“资源未释放：文件”的缺陷。

【知识点】

Java 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个文件上传代码模块，存储上传的文件，需要使用代码安全保障系统对编写的代码进行缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的缺陷检测：资源未释放：文件。

【实验原理】

Java 调用垃圾回收器来释放没有引用且没有被释放的内存。然而，垃圾回收器不能释放那些非内存资源，例如打开的文件描述符。同时虽然 Java 提供了 finalizer、PhantomReference 之类的机制来让程序向 GC 注册“自动回调释放资源”的功能。但 GC 回调它们的时机不确定，所以只应该作为最后手段来使用。如果资源未及时释放，将会降低系统性能，攻击者还可能会通过耗尽资源的方式发起拒绝服务攻击。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 2-160 所示。

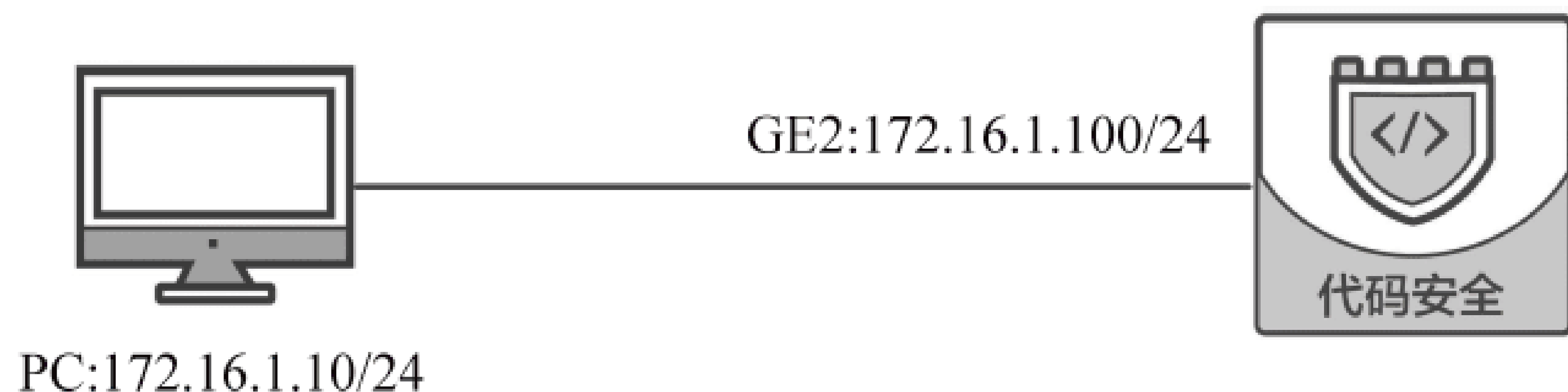


图 2-160　文件资源未释放缺陷检测实验拓扑图

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 2-161 所示。

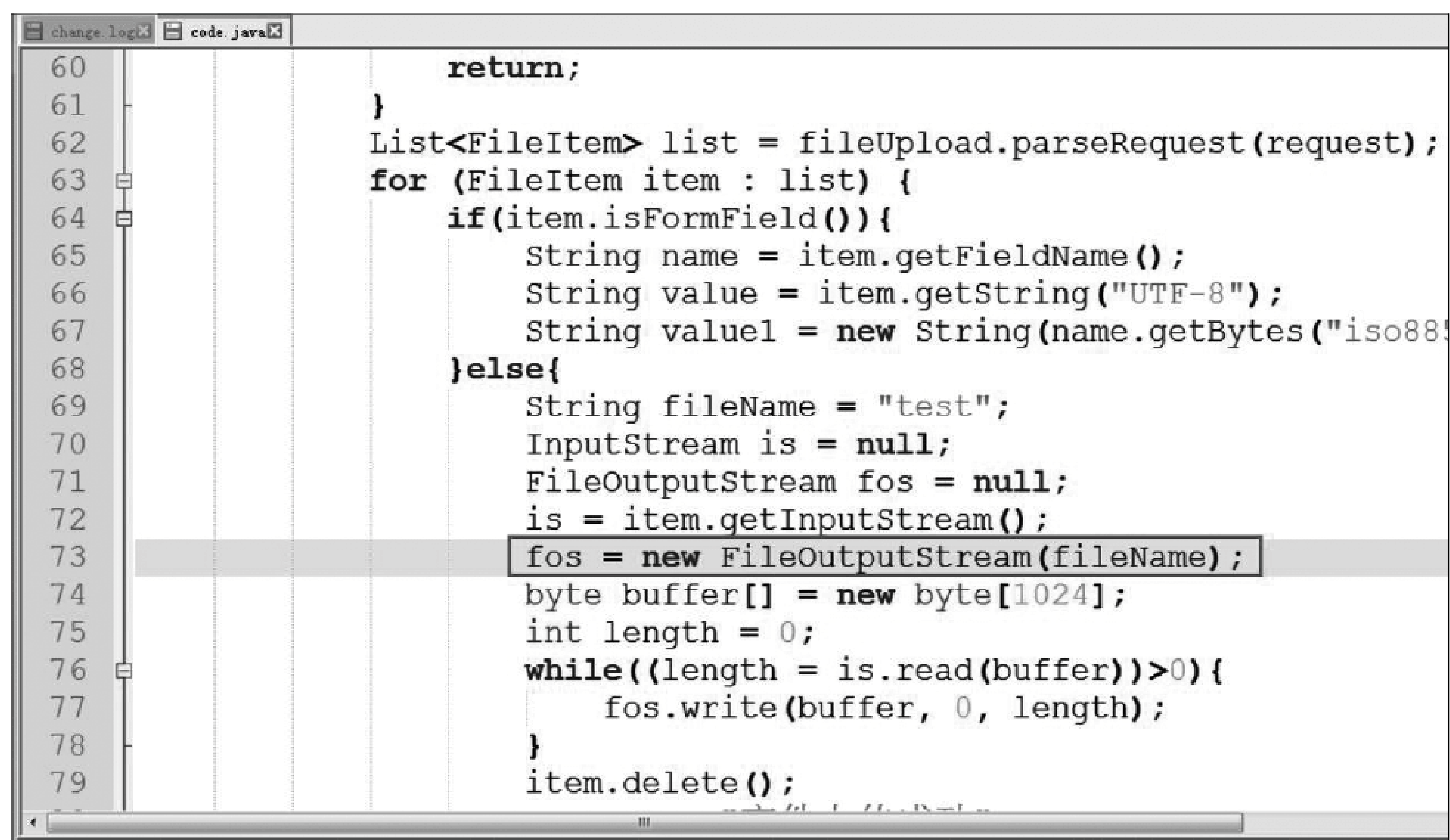


图 2-161　部分代码展示

(4) 打开 Chrome 浏览器,输入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“资源未释放-文件”,“开发语言”选中 Java 单选钮,“JDK 版本”选中“JDK1.7”单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code.zip”文件,取消勾选“合规检测”复选框,取消“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。当检测状态为“检测完成”时,单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测除了相关缺陷,并判断该缺陷的风险等级为中。单击左

侧框中的“＋”按钮，展开目录，如图 2-162 所示。



图 2-162　展开界面(2.3)

(11) 可以看到代码中存在资源未释放的位置有两处，分别为“code.java(72)”和“code.java(73)”，下面以“code.java(72)”来说明。选择“code.java(72)”命令，可以快速定位有问题的代码，如图 2-163 所示。



图 2-163　定位有问题的代码(2.3)

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。程序创建或分配流资源后，不进行合理释放，将会降低系统性能。攻击者可能会通过耗尽资源池的方式发起拒绝服务攻击，如图 2-164 所示。

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。建议表明程序不应该依赖于 finalize() 回收流资源，应在 finally 中手动释放流资源。



图 2-164　详细信息(2.3)

【实验预期】

加固后的代码再次被检测后，对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，在 finally 中手动释放资源，保存修改并关闭文件，如图 2-165 所示。



图 2-165　代码修改(2.3)

(2) 右击 code，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“资源未释放-文件修复检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中“JDK1.7”单选钮，单击“上传文件”右侧的 “浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮(图中最右框位置)，如图 2-166 所示。



图 2-166　检测完成(2.3)

(7) 可以看到缺陷已经修复，如图 2-167 所示。



图 2-167　缺陷修复

【实验思考】

(1) 如果文件资源没有手动释放会造成什么后果？

(2) Java 中有哪些资源需要手动释放？

第 3 章 代码安全保障系统合规检测

代码安全保障系统支持主流国际标准和规范的代码合规检测，包括 CERT 安全编程标准等，同时提供方便的可扩展接口，可针对组织和行业特有的代码规范，定制开发自动化检测规则，满足个性化的需求。

本章主要完成代码安全保障系统合规检测实验，根据主流编程语言将本章划分为两个模块：C/C++检测、Java 缺陷检测。通过完成这些实验，检查源代码是否违背代码开发规范，约束并规范开发者的开发行为。规范的代码可以有效减少 Bug 处理、降低代码维护成本以及帮助程序员进行代码审查。

3.1 C/C++合规检测

3.1.1 对环境变量的长度进行假设合规检测实验

【实验目的】

确保所编写的代码中不存在对环境变量长度进行假设的情况。

【知识点】

“对环境变量长度进行假设”违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个系统信息查询模块，通过调用库函数来获取系统环境变量，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要对环境变量的长度进行假设。

【实验原理】

环境变量一般是指在操作系统中用来指定操作系统运行环境的一些参数。它的长度是不固定的。如果在对其进行复制时，对其长度进行假设，很容易造成缓冲区溢出，这是十分危险的。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-1 所示。

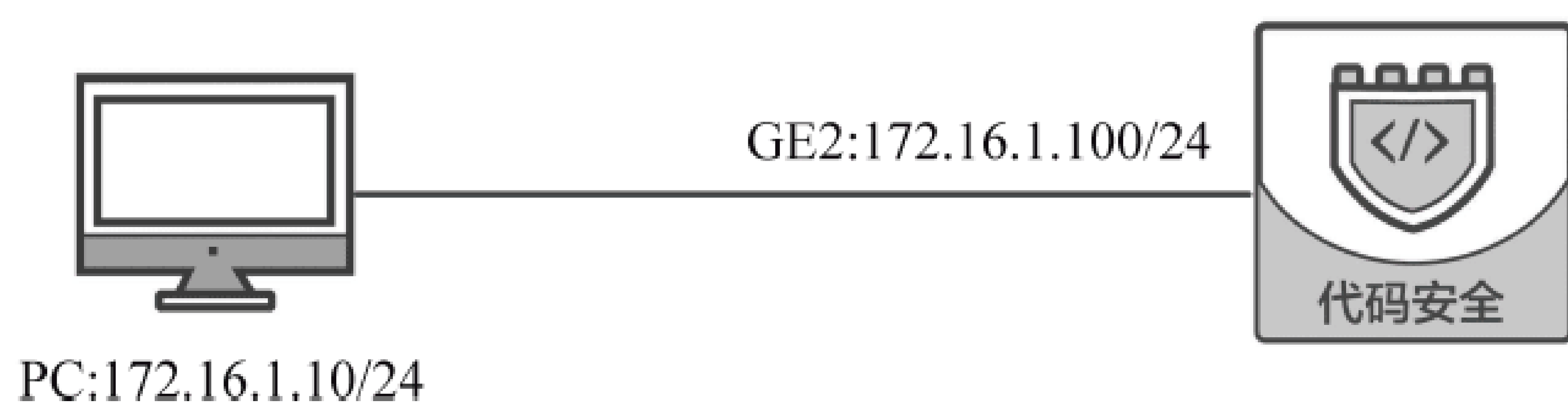


图 3-1　对环境变量的长度进行假设合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选项选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.c，单击“添加”按钮。

(7) 打开“C:\ ”下的 code.txt，复制其中的代码。

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资源管理器”按钮窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“预处理器”选项。

(11) 单击“预处理器定义”按钮。

(12) 单击右方的“下拉”按钮。

(13) 选择"编辑"命令。

(14) 在"预处理器定义"中输入"_CRT_SECURE_NO_WARNINGS"后单击"确定"按钮。

(15) 单击"确定"按钮。

(16) 单击"本地 Windows 调试器"按钮,如图 3-2 所示。

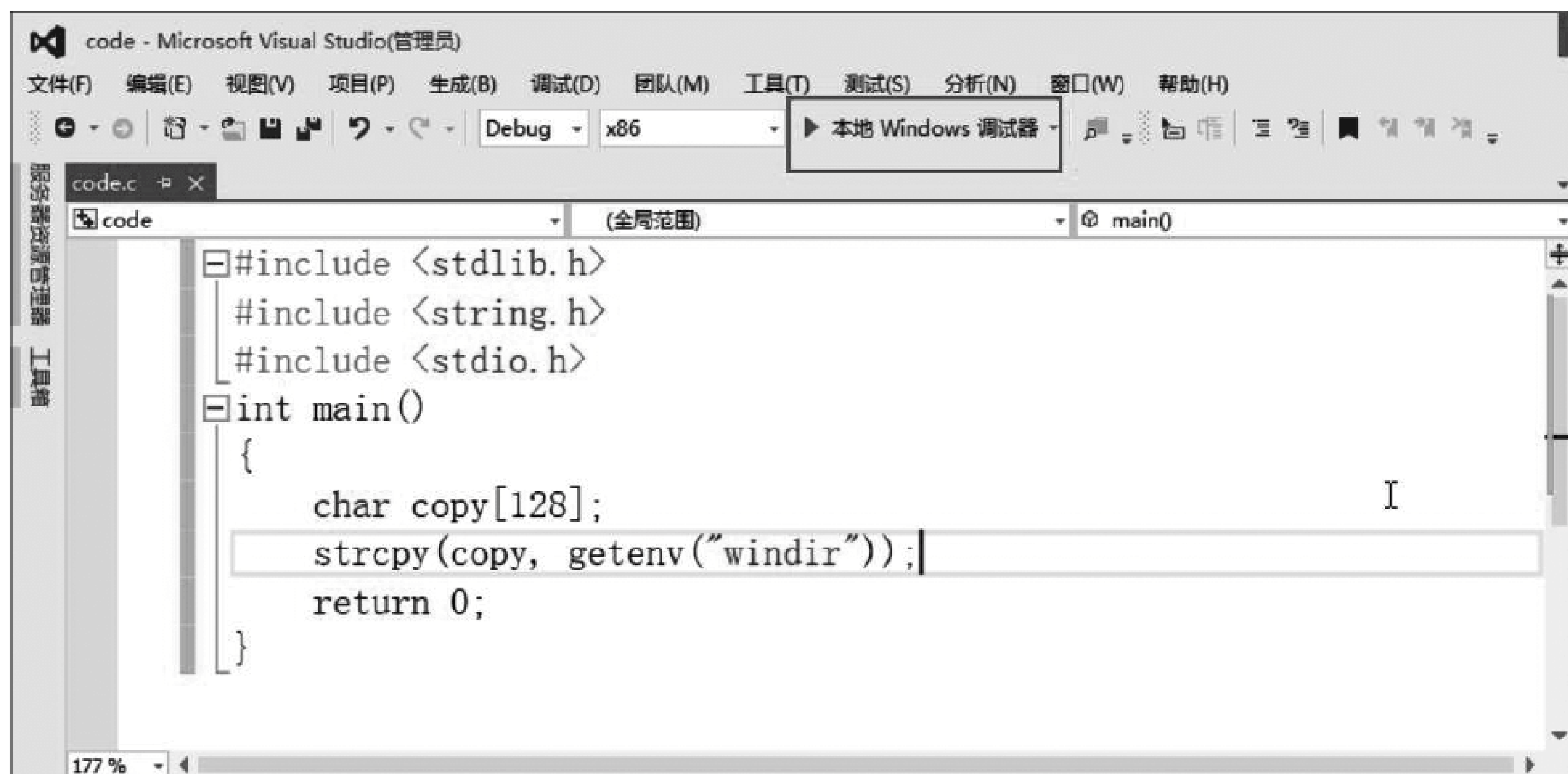


图 3-2 代码编辑界面(3.1.1)

(17) 弹出"编译确认"对话框,单击"是"按钮。

(18) 选择"开始"→"VS 2015 开发人员命令提示"命令,打开命令行界面。

(19) 进入"管理员:VS 2015 开发人员命令提示"界面,在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(20) 进入终端机"C:\ "下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器,网址输入"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(22) 在"高级"界面中单击"继续前往 172.16.1.100(不安全)"按钮。

(23) 界面跳转到代码卫士登录界面,"用户名"输入 admin,"密码"输入"admin123!@#",单击"登录"按钮。

(24) 选择"快速检测"→"+发起快速检测"命令。

(25) "任务名称"输入"不要对环境变量的长度进行假设","开发语言"选中"C/C++"单选钮,勾选"合规检测模板[C/C++]"复选框。

(26) 单击"浏览"按钮。

(27) 单击"本地磁盘(C:)",选择右侧的 codemanager,单击"打开"按钮。

(28) 选择文件 codemanager,单击"打开"按钮。

(29) 返回"快速检测"界面,单击"发起检测"按钮。

【实验预期】

(1) 检测出代码中对环境变量的长度进行假设的违规。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 查看检测结果,修改代码

(1) 在终端机打开 Chrome 浏览器,在地址栏中输入"https://172.16.1.100",进入代码安全保障系统登录界面。输入用户名 admin,密码为"admin123!@#",单击"登录"按钮。

(2) 选择"快速检测"命令。

(3) 选择"合规检测"命令,查看列表所示检测结果,如图 3-3 所示。



图 3-3　"合规检测"界面(3.1.1)

(4) 找到"不要对环境变量的长度进行假设",单击右侧的"缺陷审计"按钮。

(5) 在左侧查看代码的缺陷,如图 3-4 所示。

(6) 打开"C:\code"文件夹,双击 code.sln,打开项目。

(7) 在代码编辑界面修改代码,单击"本地 Windows 调试器"按钮,如图 3-5 所示。

(8) 打开"C:\codemanager"文件夹,删除其中的所有文件。

(9) 选择"开始"→"VS 2015 开发人员命令提示"命令,打开命令行界面。

(10) 进入"管理员: VS 2015 开发人员命令提示"界面,在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(11) 进入终端机"C:\ "下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(12) 进入 Chrome 浏览器代码卫士界面,选择"快速检测"→"+发起快速检测"命令。

(13) "任务名称"输入"不要对环境变量的长度进行假设","开发语言"选中"C/C++"



图 3-4 “快速检测”界面(3.1.1)

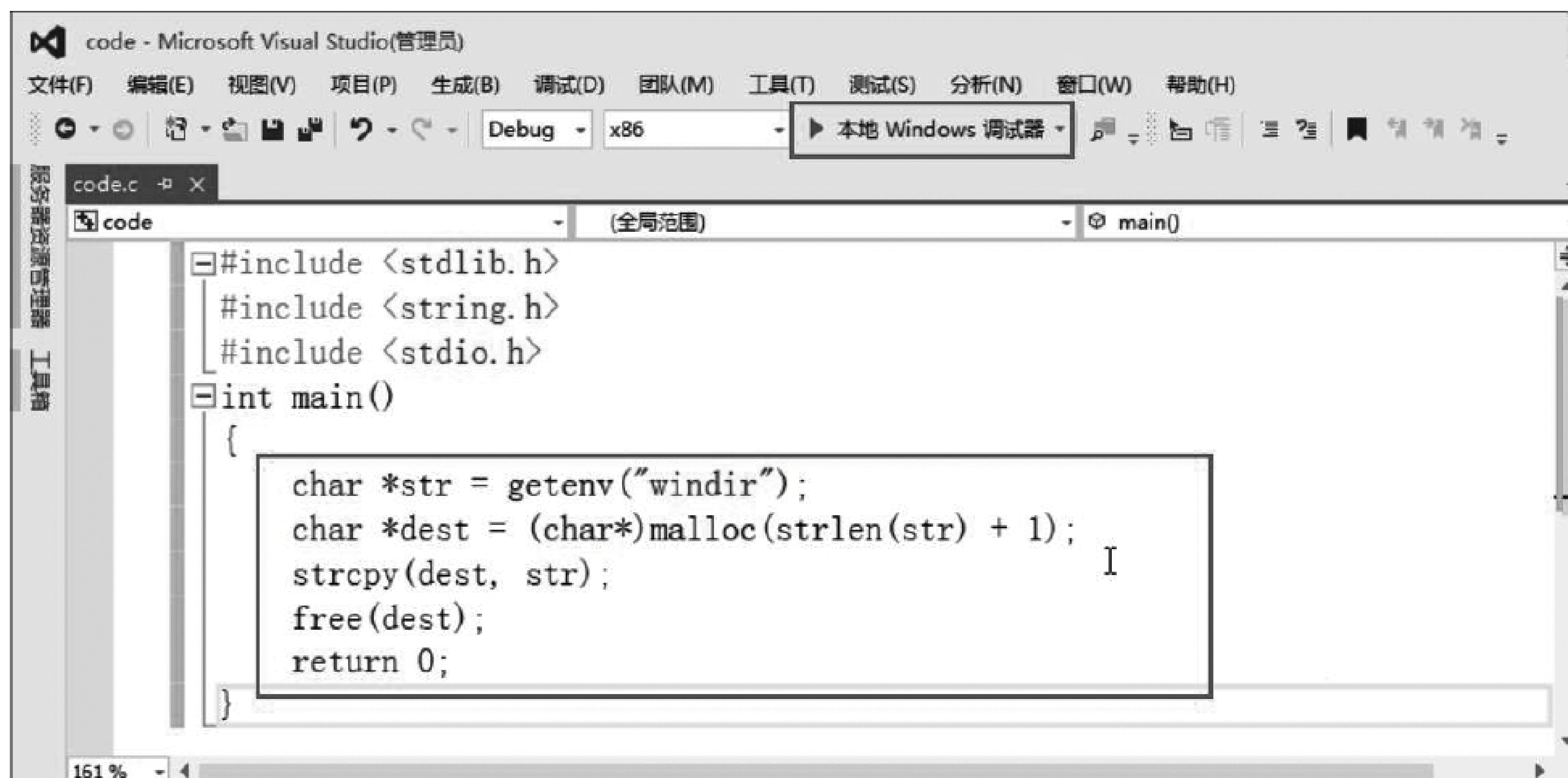


图 3-5 修改代码(3.1.1)

单选钮，勾选“合规检测模板[C/C++]”复选框。

(14) 单击“浏览”按钮。

(15) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(16) 选择文件 codemanager，单击“打开”按钮。

(17) 返回“快速检测”界面，单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 3-6 所示。



图 3-6　查看结果(3.1.1)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考如何使用代码修改环境变量。

3.1.2　检测并处理库函数中的错误合规检测实验

【实验目的】

确保所编写的代码中不存在没有检测处理库函数中的错误的情况。

【知识点】

“未检测或处理库函数中的错误”违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个库函数封装模块，对库函数进行了封装调用，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：检测并处理库函数中的错误。

【实验原理】

C 语言中有许多库函数，这些库函数在失败时会返回错误信息，如果不加判断就默认函数成功返回，那么在程序执行过程中就无法对库函数调用异常、错误的情况进行有效的处理，进而可能出现错误导致程序崩溃。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑图如图 3-7 所示。

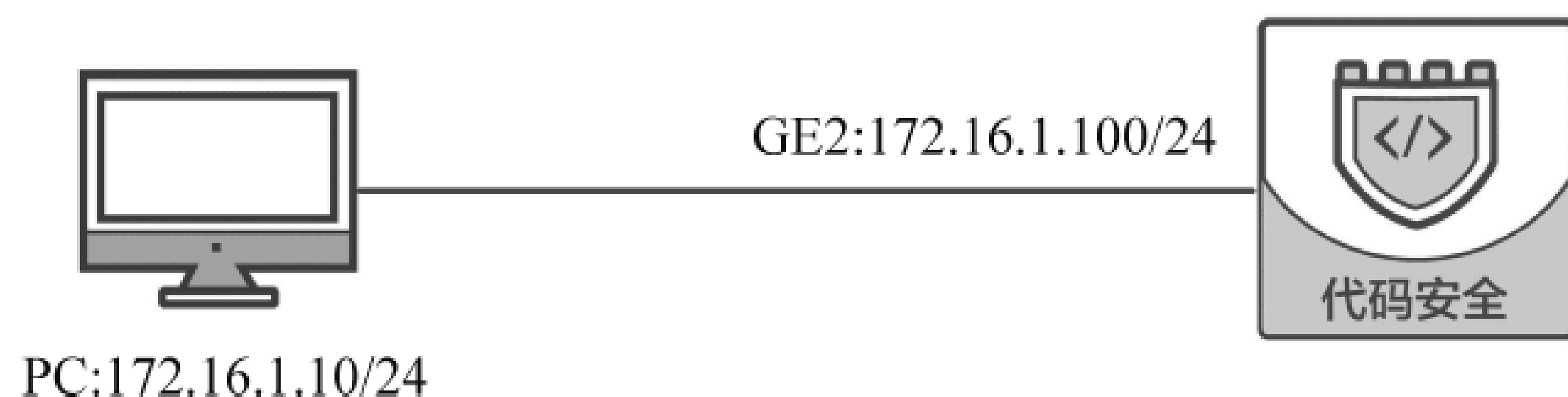


图 3-7　检测并处理库函数中的错误合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士,检测出缺陷。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456,如图 3-8 所示。

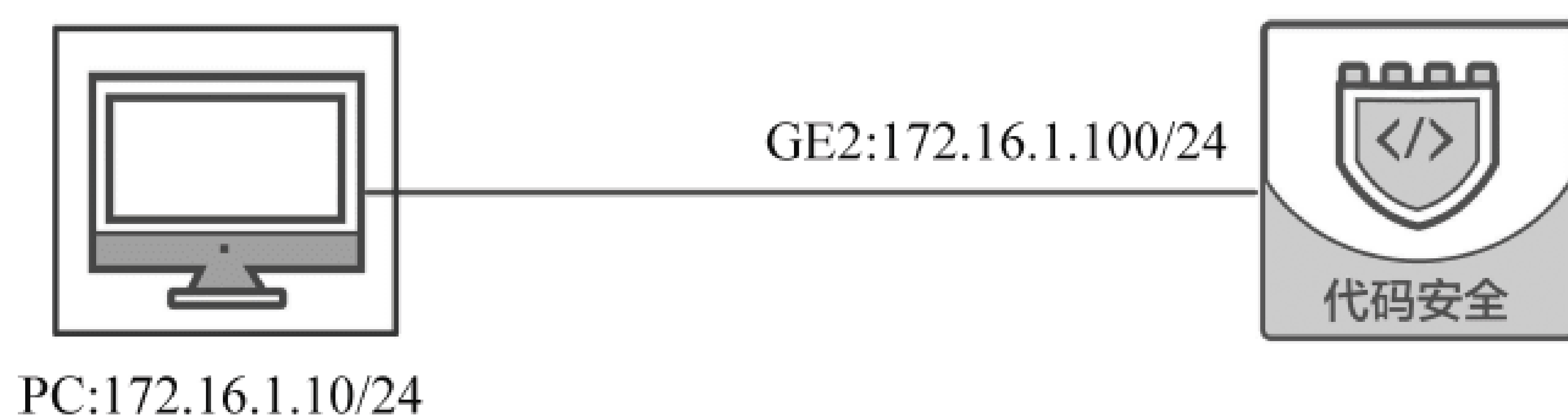


图 3-8　对应实验平台拓扑图

(2) 在学生机桌面双击 Visual Studio 2015,显示主界面。

(3) 进入主界面,选择“文件”→“新建”→“项目”命令,新建一个项目。

(4) 进入“新建项目”界面,在左侧“模板”选择“Visual C++”命令,界面中间选项选择“空项目”,下侧“名称”选项框内输入 code,单击“位置”右侧的“浏览”按钮,选择“C:\”,其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上“解决方案资源管理器”小窗口,已经新建好 code 项目。右击“源文件”,在弹出的快捷菜单中选择“添加”→“新建项”命令,添加一个新项。

(6) 进入“添加新项”界面,左侧“已安装”默认选择“Visual C++”命令,在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.c,单击“添加”按钮。

(7) 打开“C:\”下的 code.txt,复制其中的代码。

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“预处理器”选项。

(11) 选择“预处理器定义”命令。

(12) 单击右方的“下拉”按钮。

(13) 选择“编辑”命令。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框,单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(19) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入学生机“C:\ ”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”, 在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(22) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(23) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(24) 选择“快速检测”→“+发起快速检测”命令。

(25) “任务名称”输入“检测并处理库函数中的错误”,“开发语言”选中“C/C++”单选钮,勾选“合规检测模板[C/C++]”复选框。

(26) 单击“浏览”按钮。

(27) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(28) 选择文件 codemanager,单击“打开”按钮。

(29) 返回“快速检测”界面,单击“发起检测”按钮。

【实验预期】

(1) 查看检测结果,修改代码。

(2) 修改代码,对比修改前后检测结果。

【实验结果】

1. 查看检测结果,修改代码

(1) 在学生本地机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 选择“快速检测”命令。

(3) 选择“合规检测”命令,查看列表所示检测结果。

(4) 找到“检测并处理库函数中的错误”,单击右侧的“缺陷审计”按钮。

(5) 在左侧查看代码的缺陷,如图 3-9 所示。



图 3-9 “快速检测”界面(3.1.2)

(6) 打开“C:\code”文件夹，双击 code.sln，打开项目。

(7) 在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，运行结束后关闭 VS 2015。

(8) 打开“C:\codemanager”文件夹，删除其中的所有文件然后关闭。

(9) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(10) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(11) 进入学生机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(12) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“+发起快速检测”命令。

(13) “任务名称”输入“检测并处理库函数中的错误”，“开发语言”选中“C/C++”单选钮，勾选“合规检测模板[C/C++]”复选框。

(14) 单击“浏览”按钮。

(15) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(16) 选择文件 codemanager，单击“打开”按钮。

(17) 返回“快速检测”界面，单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 3-10 所示。

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考像 malloc 这样需要检测错误的常用的库函数还有哪些。



图 3-10　查看结果(3.1.2)

3.1.3　较大长度的值比较或赋值合规检测实验

【实验目的】

确保所编写的代码中在比较或赋值给一个较大长度的值之前已将整型表达式的值转换为这个较大长度。

【知识点】

“在比较或赋值给一个较大长度的值之前未将整型表达式的值转换为这个较大长度”违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个整型运算模块，对传入的整型数据进行数学运算，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度。

【实验原理】

在 C 语言中，不同类型的数值在进行比较或赋值时，会进行强制类型转化，这种转化有时候并不是保值的。如果在比较或赋值前没有将整型表达式的值转换成较长长度，则会使比较或赋值结果出现问题。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行有针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。

• 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-11 所示。

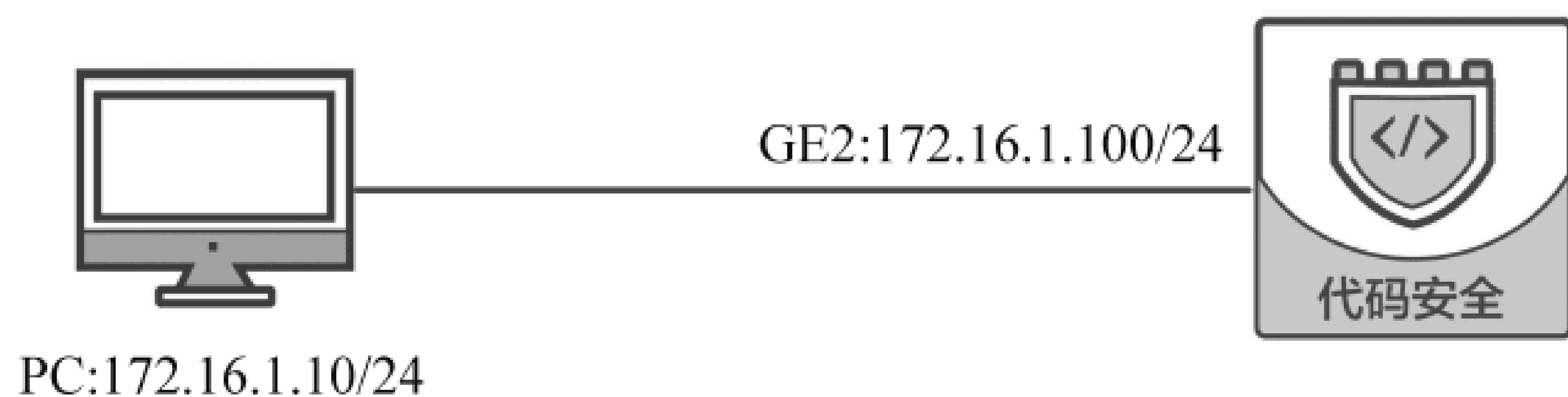


图 3-11　较大长度的值比较或赋值合规检测实验拓扑图

【实验思路】

（1）建立项目文件。

（2）生成代码的中间表示文件。

（3）上传到代码卫士，检测出缺陷。

（4）依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）在学生机桌面双击 Visual Studio 2015，显示主界面。

（3）进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

（4）进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

（5）再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

（6）进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.c，单击“添加”按钮。

（7）打开“C:\”下的 code.txt，复制其中的代码，如图 3-12 所示。

（8）进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资源管理器”窗口中的 code 项目。

（9）选择“属性”命令。

（10）选择“配置属性”中的“C/C++”下的“预处理器”选项。

（11）选择“预处理器定义”命令。

（12）单击右方的“下拉”按钮。

（13）选择“编辑”命令。

（14）在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮。

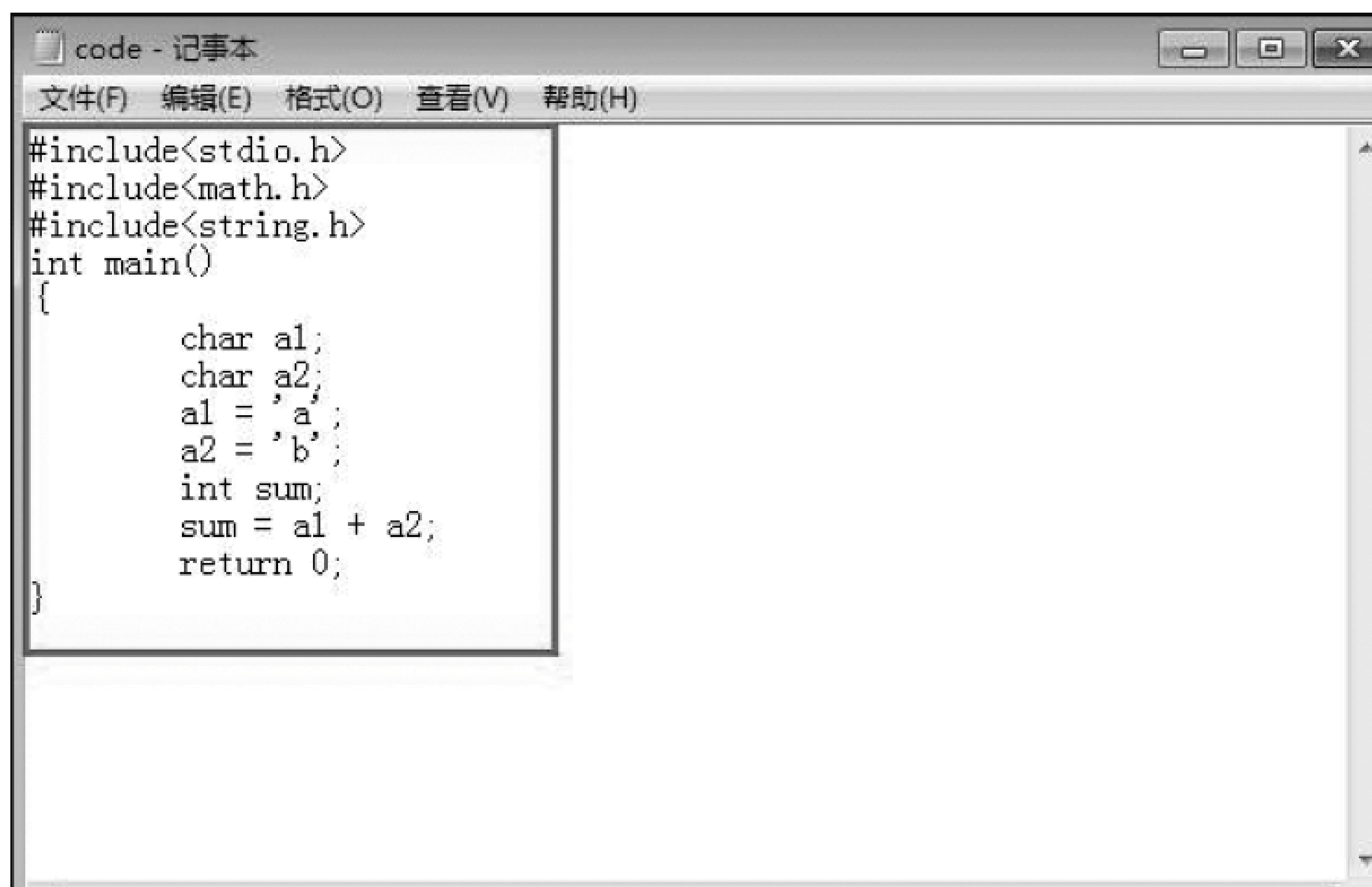


图 3-12　复制代码(3.1.3)

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框，单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(19) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入学生机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(22) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(23) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(24) 选择“快速检测”→“+发起快速检测”命令。

(25) “任务名称”输入“在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度”，“开发语言”选中“C/C++”单选钮，勾选“合规检测模板[C/C++]”复选框。

(26) 单击“浏览”按钮。

(27) 单击“本地磁盘(C:)”按钮，选择右侧的 codemanager，单击“打开”按钮。

(28) 选择文件 codemanager，单击“打开”按钮。

(29) 返回“快速检测”界面，单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中在比较或赋值给一个较大长度的值之前未将整型表达式的值转换为这个较大长度的违规。

（2）给出该缺陷的详细描述和修复建议。

【实验结果】

1. 查看检测结果，修改代码

（1）在终端机打开 Chrome 浏览器，在地址栏中输入“https://172.16.1.100”，进入代码安全保障系统登录界面。输入用户名 admin，密码为“admin123!@＃”，单击“登录”按钮。

（2）选择“快速检测”命令。

（3）选择“合规检测”命令，查看列表所示检测结果。

（4）找到“在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度”，单击右侧的“缺陷审计”按钮。

（5）在左侧查看代码的缺陷，如图 3-13 所示。



图 3-13 “快速检测”界面(3.1.3)

（6）打开“C:\code”文件夹，双击 code.sln 打开项目。

（7）在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 3-14 所示。

（8）打开“C:\codemanager”文件夹，删除其中的所有文件然后关闭。

（9）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

（10）进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

（11）进入学生机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

（12）进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”命令。

（13）“任务名称”输入“在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度”，“开发语言”选中“C/C++”单选钮，勾选“合规检测模板[C/C++]”复选框。

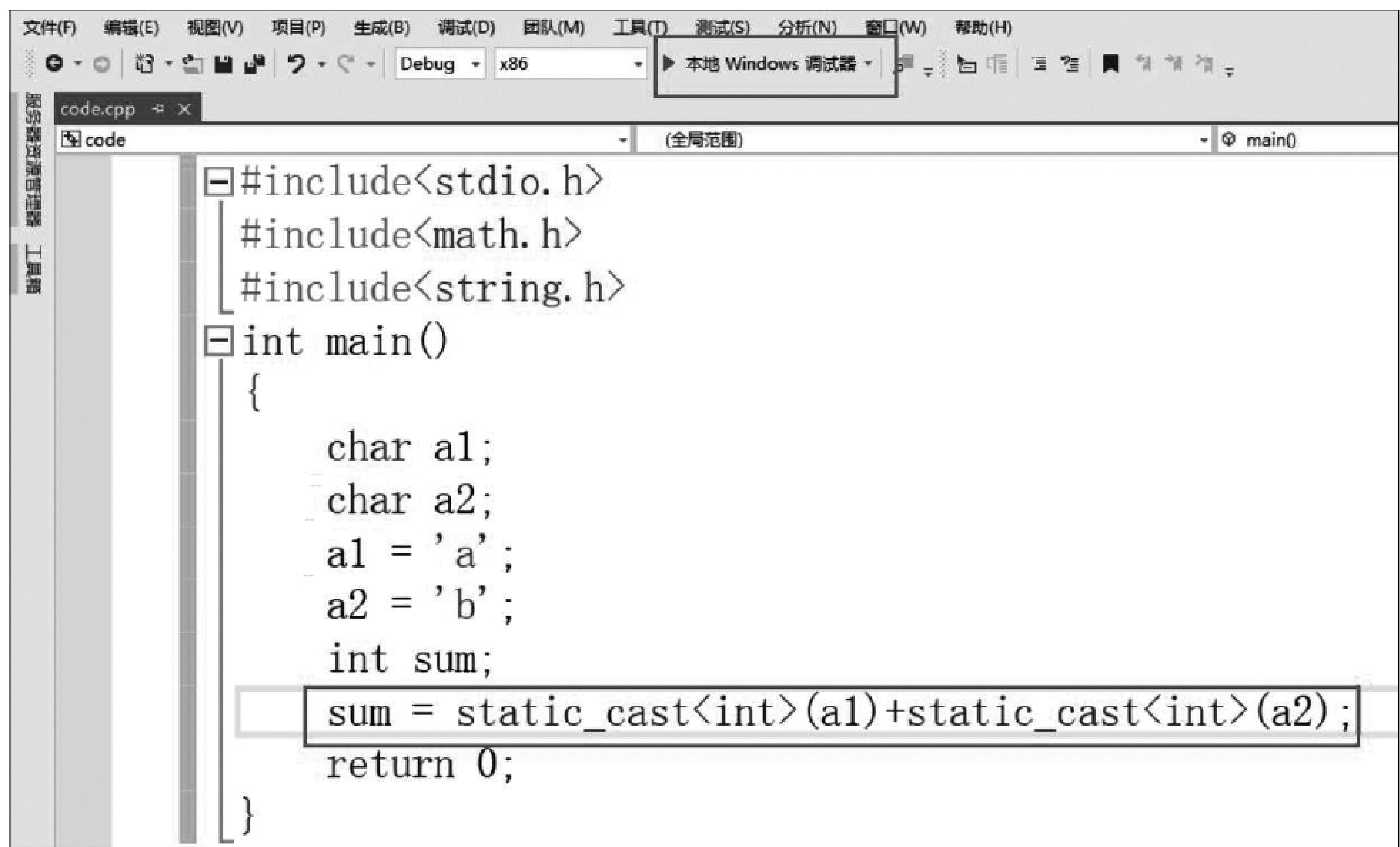


图 3-14　修改代码(3.1.3)

(14) 选择“浏览”命令。

(15) 选择“本地磁盘(C:)”命令，选择右侧的 codemanager，单击“打开”按钮。

(16) 选择文件 codemanager，单击“打开”按钮。

(17) 返回“快速检测”界面，单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 3-15 所示。



图 3-15　查看结果(3.1.3)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考如果在比较或赋值给一个较大长度的值之前没有将整型表达式的值转换为这个较大长度结果会怎样。

3.1.4 代码在 free( )之后立即在指针中存储一个新值合规检测实验

【实验目的】

确保所编写的代码中删除在 free()之后立即在指针中存储一个新值的情况。

【知识点】

"在 free()之后立即在指针中存储一个新值"违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个内存处理模块，调用 malloc/free 库函数为指针反复申请/释放内存，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：在 free()之后立即在指针中存储一个新值。

【实验原理】

在 C 语言中，利用 malloc()函数可以为指针动态分配内存。分配的动态内存在不需要的时候需要用 free()函数进行释放，在释放后，指针如果需要重复使用，则需要再次利用 malloc()函数进行内存分配，不能直接对其赋值。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-16 所示。

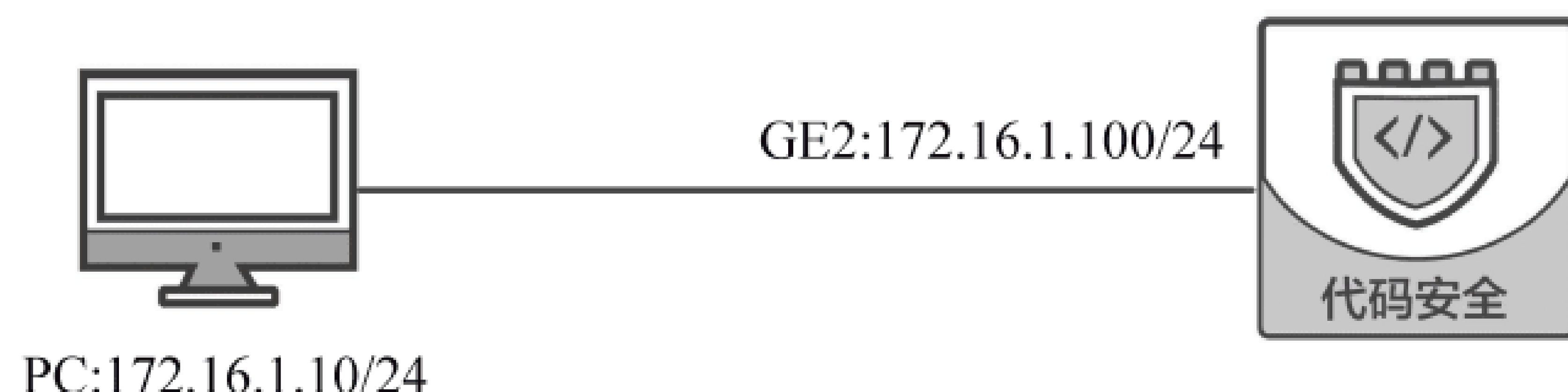


图 3-16 代码在 free()之后立即在指针中存储一个新值合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士,检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 在学生机桌面双击 Visual Studio 2015,显示主界面。

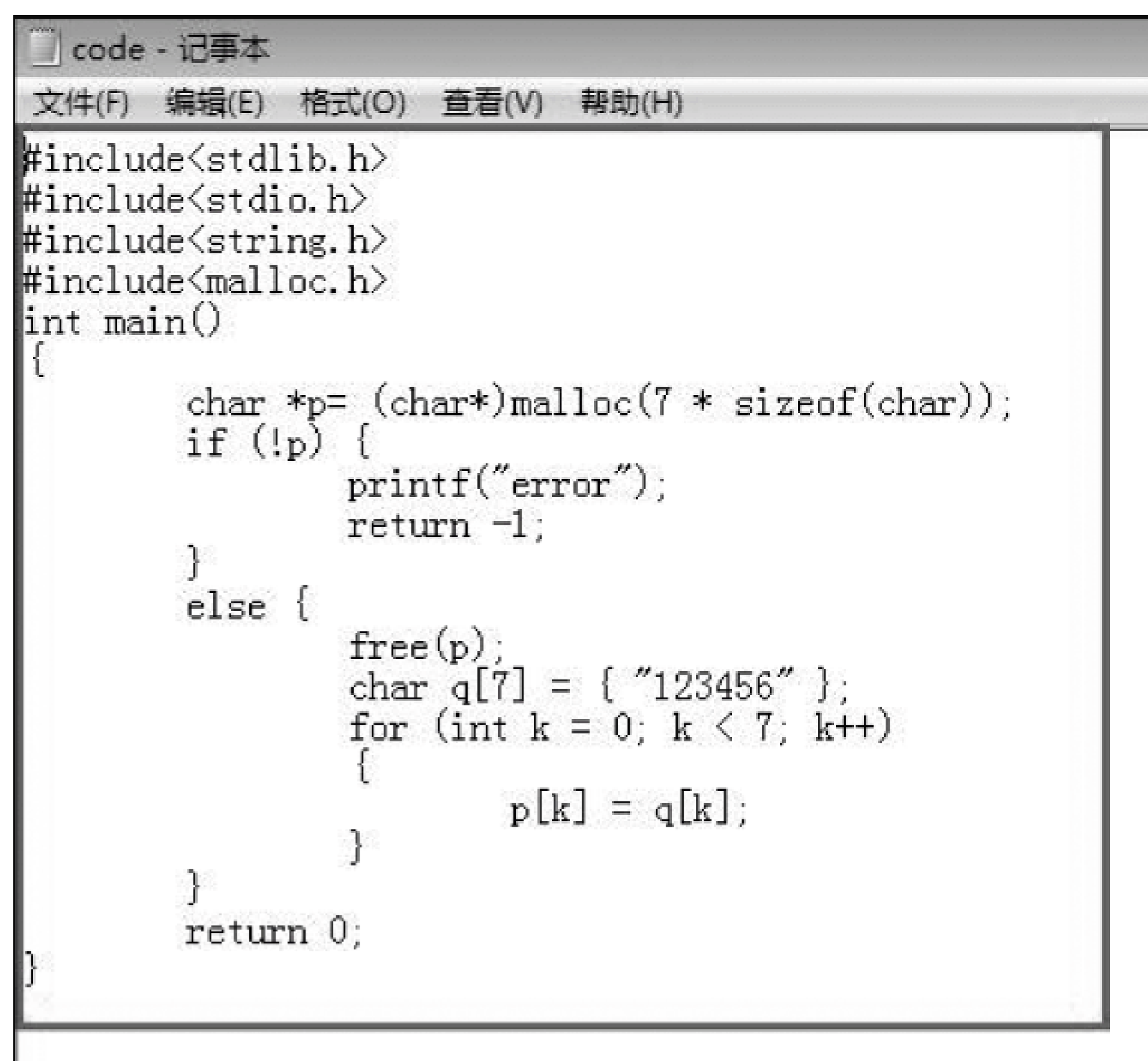
(3) 进入主界面,选择“文件”→“新建”→“项目”命令,新建一个项目。

(4) 进入“新建项目”界面,在左侧“模板”选择“Visual C++”命令,界面中间选项选择“空项目”,下侧“名称”选项框内输入 code,单击“位置”右侧的“浏览”按钮,选择“C:\”,其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上“解决方案资源管理器”小窗口,已经新建好 code 项目。右击“源文件”,在弹出的快捷菜单中选择“添加”→“新建项”命令,添加一个新项。

(6) 进入“添加新项”界面,左侧“已安装”默认选择“Visual C++”命令,在界面中间选择“C++文件(.cpp)”。在下侧“名称”选项框内输入 code.c,单击“添加”按钮。

(7) 打开“C:\”下的 code.txt,复制其中的代码,如图 3-17 所示。



code - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<malloc.h>
int main()
{
	char *p= (char*)malloc(7 * sizeof(char));
	if (!p) {
		printf("error");
		return -1;
	}
	else {
		free(p);
		char q[7] = { "123456" };
		for (int k = 0; k < 7; k++)
		{
			p[k] = q[k];
		}
	}
	return 0;
}
```

图 3-17　复制代码(3.1.4)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“预处理器”选项。

(11) 选择“预处理器定义”命令。

(12) 单击右方的“下拉”按钮。

(13) 选择“编辑”命令。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框,单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(19) 进入“管理员:VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入学生机“C:\ ”下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”, 在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(22) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(23) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(24) 选择“快速检测”→“+发起快速检测”命令。

(25) “任务名称”输入“在比较或赋值给一个较大长度的值之前在 free 之后立即在指针中存储一个新值”,“开发语言”选中“C/C++”单选钮,勾选“合规检测模板[C/C++]”复选框。

(26) 选择“浏览”命令。

(27) 选择“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(28) 选择文件 codemanager,单击“打开”按钮。

(29) 返回“快速检测”界面,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中在 free()之后立即在指针中存储一个新值的违规。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 查看检测结果,修改代码

(1) 在终端机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 选择“快速检测”命令。

(3) 选择“合规检测”命令,查看列表所示检测结果。

（4）找到“在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度”，单击右侧的“缺陷审计”按钮。

（5）在左侧查看代码的缺陷，如图 3-18 所示。



图 3-18　“快速检测”界面(3.1.4)

（6）打开“C:\code”文件夹，双击 code.sln，打开项目。

（7）在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 3-19 所示。

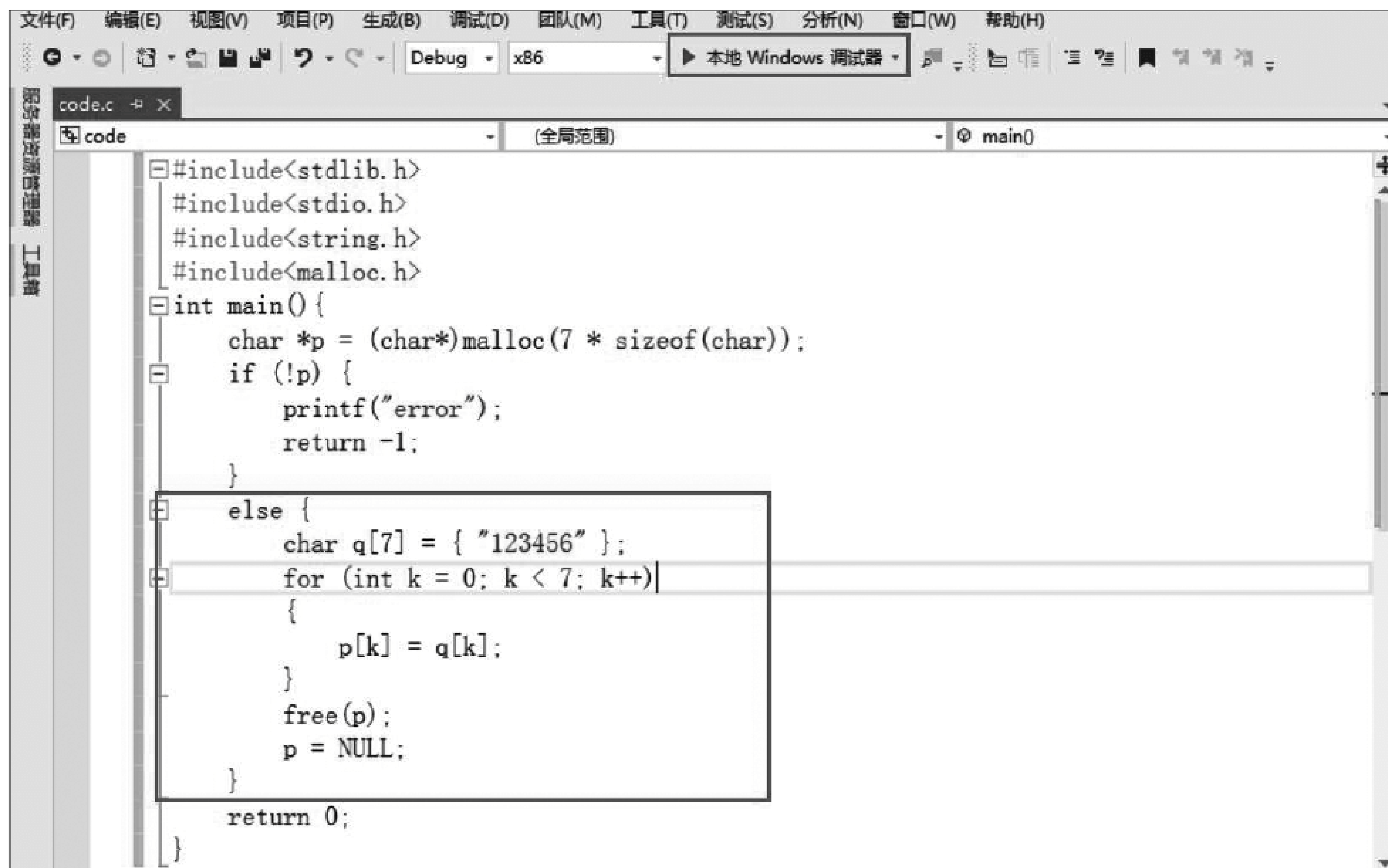


图 3-19　修改代码(3.1.4)

（8）打开“C:\codemanager”文件夹，删除其中的所有文件然后关闭。

（9）选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(10) 进入"管理员：VS 2015 开发人员命令提示"界面，在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(11) 进入学生机"C:\ "下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(12) 进入 Chrome 浏览器代码卫士界面，选择"快速检测"→"＋发起快速检测"命令。

(13) "任务名称"输入"在 free 之后立即在指针中存储一个新值"，"开发语言"选中"C/C++"单选钮，勾选"合规检测模板[C/C++]"复选框。

(14) 选择"浏览"命令。

(15) 选择"本地磁盘(C:)"命令，选择右侧的 codemanager，单击"打开"按钮。

(16) 选择文件 codemanager，单击"打开"按钮。

(17) 返回"快速检测"界面，单击"发起检测"按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后，单击任务右侧的"缺陷审计"按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 3-20 所示。



图 3-20 查看结果(3.1.4)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考 free 之后的指针指向哪里。

3.1.5 代码只释放动态分配的内存合规检测实验

【实验目的】

确保所编写的代码中只释放动态分配的内存。

【知识点】

“释放非动态分配的内存”违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个内存处理模块，对内存进行分配与释放操作，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：只释放动态分配的内存。

【实验原理】

C/C++中 free 函数与 delete 操作符只能分别释放由 malloc 函数和 new 操作符申请的动态内存，如果释放其他内存则会出现错误。

代码安全保障系统能够检测出代码中释放非动态分配的内存的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-21 所示。

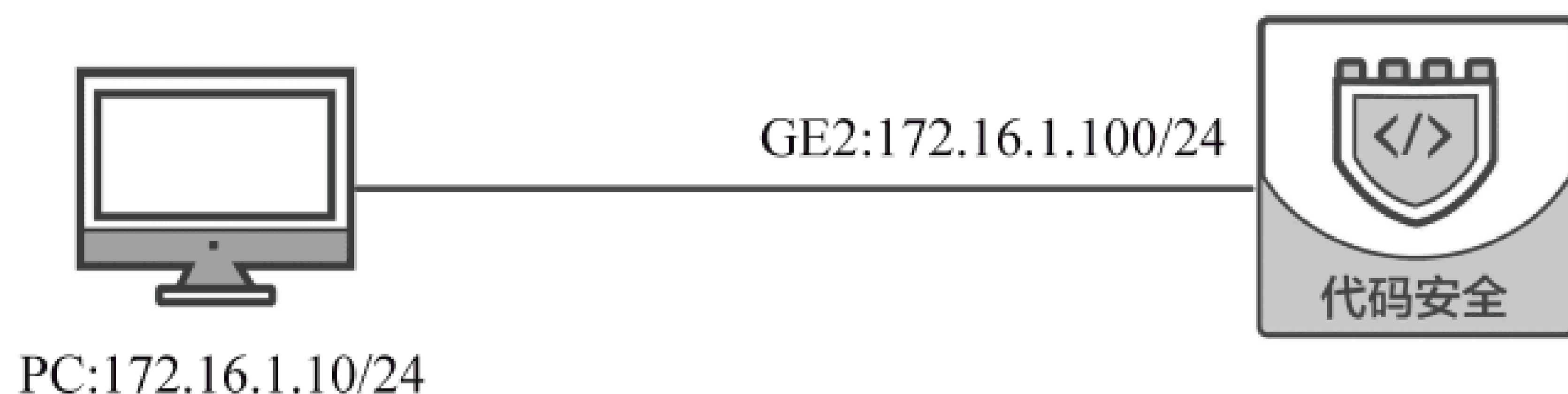


图 3-21　代码只释放动态分配的内存合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在学生机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

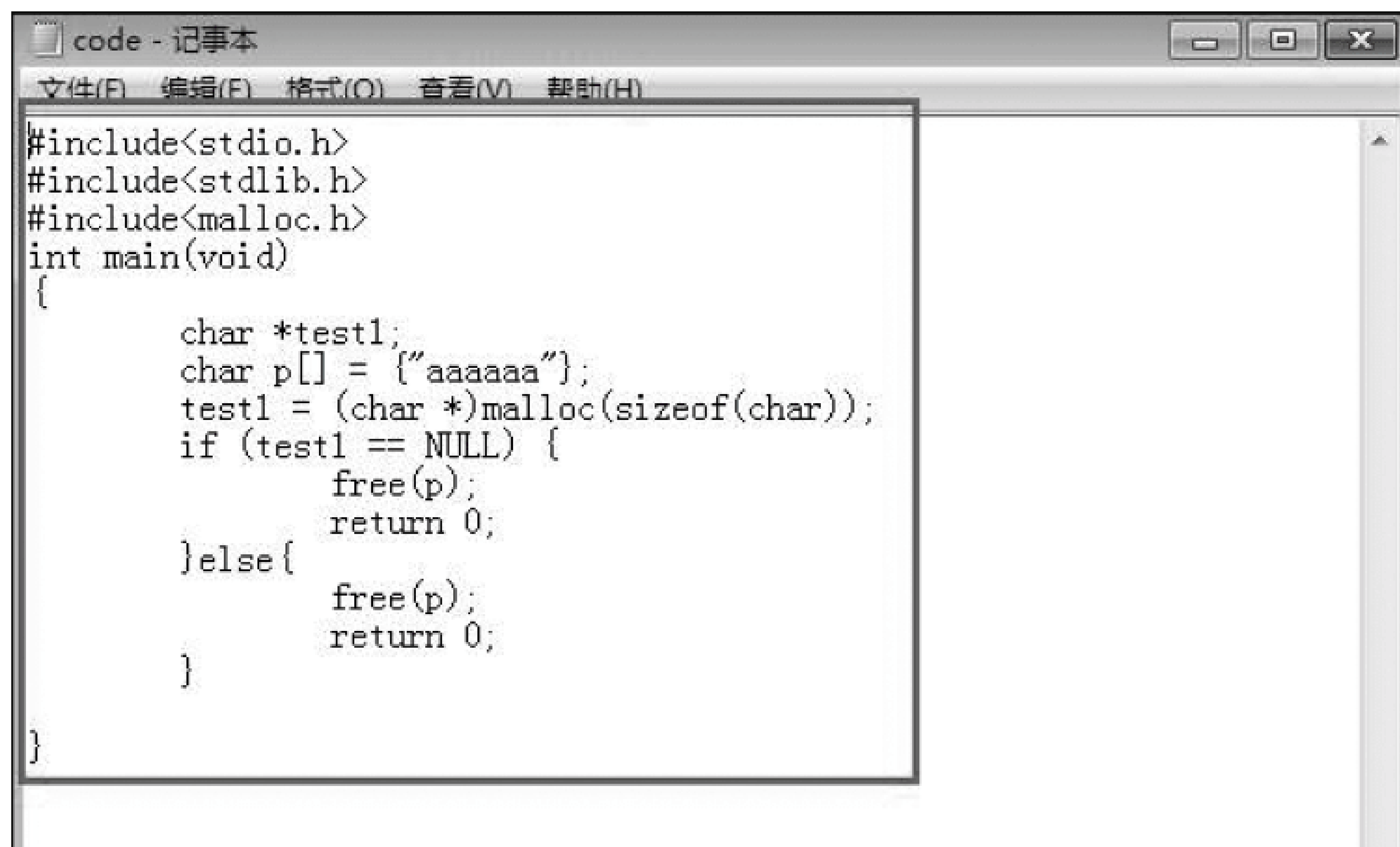
(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”右侧的“浏览”按钮，选择“C:\”，其

他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”按钮。在下侧“名称”选项框内输入 code.c，单击“添加”按钮。

(7) 打开“C:\”下的 code.txt，复制其中的代码，如图 3-22 所示。




```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
int main(void)
{
        char *test1;
        char p[] = {"aaaaaa"};
        test1 = (char *)malloc(sizeof(char));
        if (test1 == NULL) {
                free(p);
                return 0;
        }else{
                free(p);
                return 0;
        }

}
```

图 3-22 复制代码(3.1.5)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“预处理器”选项。

(11) 选择“预处理器定义”命令。

(12) 单击右方的“下拉”按钮。

(13) 选择“编辑”命令。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框，单击“是”按钮。

(18) 由于使用 free 函数释放了非动态创建的内存，在运行时会出现如下错误，单击“中止”按钮，如图 3-23 所示。

(19) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(20) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(21) 进入学生机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip

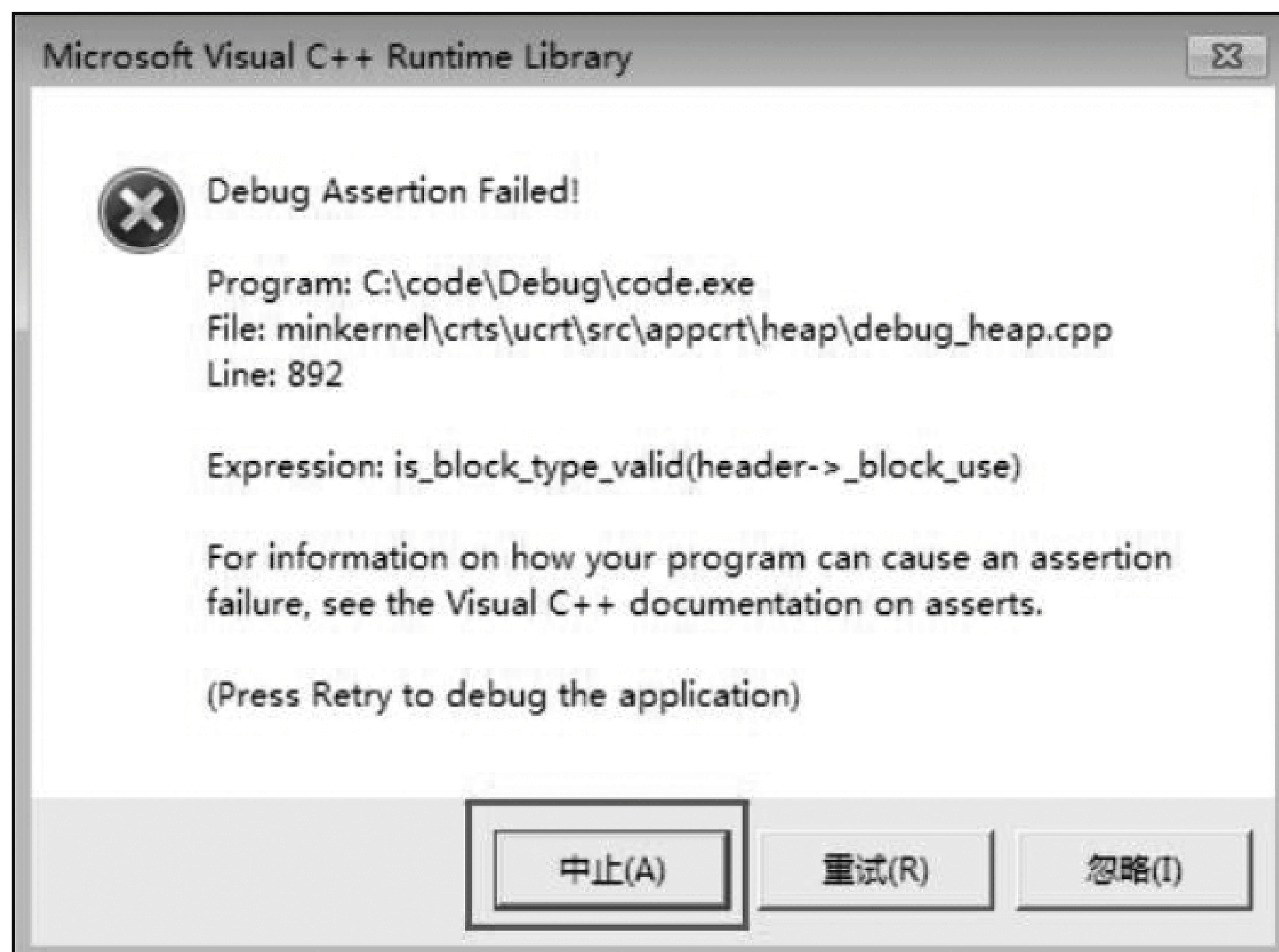


图 3-23　运行错误(3.1.5)

生成成功。

(22) 打开 Chrome 浏览器,网址输入“https://172.16.1.100”,在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(23) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(24) 界面跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(25) 选择“快速检测”→“+发起快速检测”命令。

(26) “任务名称”输入“只释放动态分配的内存”,“开发语言”选中“C/C++”单选钮,勾选“合规检测模板[C/C++]”复选框。

(27) 选择“浏览”命令。

(28) 选择“本地磁盘(C:)”命令,选择右侧的 codemanager,单击“打开”按钮。

(29) 选择文件 codemanager,单击“打开”按钮。

(30) 返回“快速检测”界面,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放非动态分配的内存的违规。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 查看检测结果,修改代码

(1) 在终端机打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@#”,单击“登录”按钮。

(2) 选择“快速检测”命令。

(3) 选择“合规检测”命令，查看列表所示检测结果。

(4) 找到“在比较或赋值给一个较大长度的值之前将整型表达式的值转换为这个较大长度”，单击右侧的“缺陷审计”按钮。

(5) 在左侧查看代码的缺陷，如图 3-24 所示。



图 3-24 “快速检测”界面(3.1.5)

(6) 打开“C:\code”文件夹，双击 code.sln，打开项目。

(7) 在代码编辑界面修改代码，单击“本地 Windows 调试器”按钮，如图 3-25 所示。

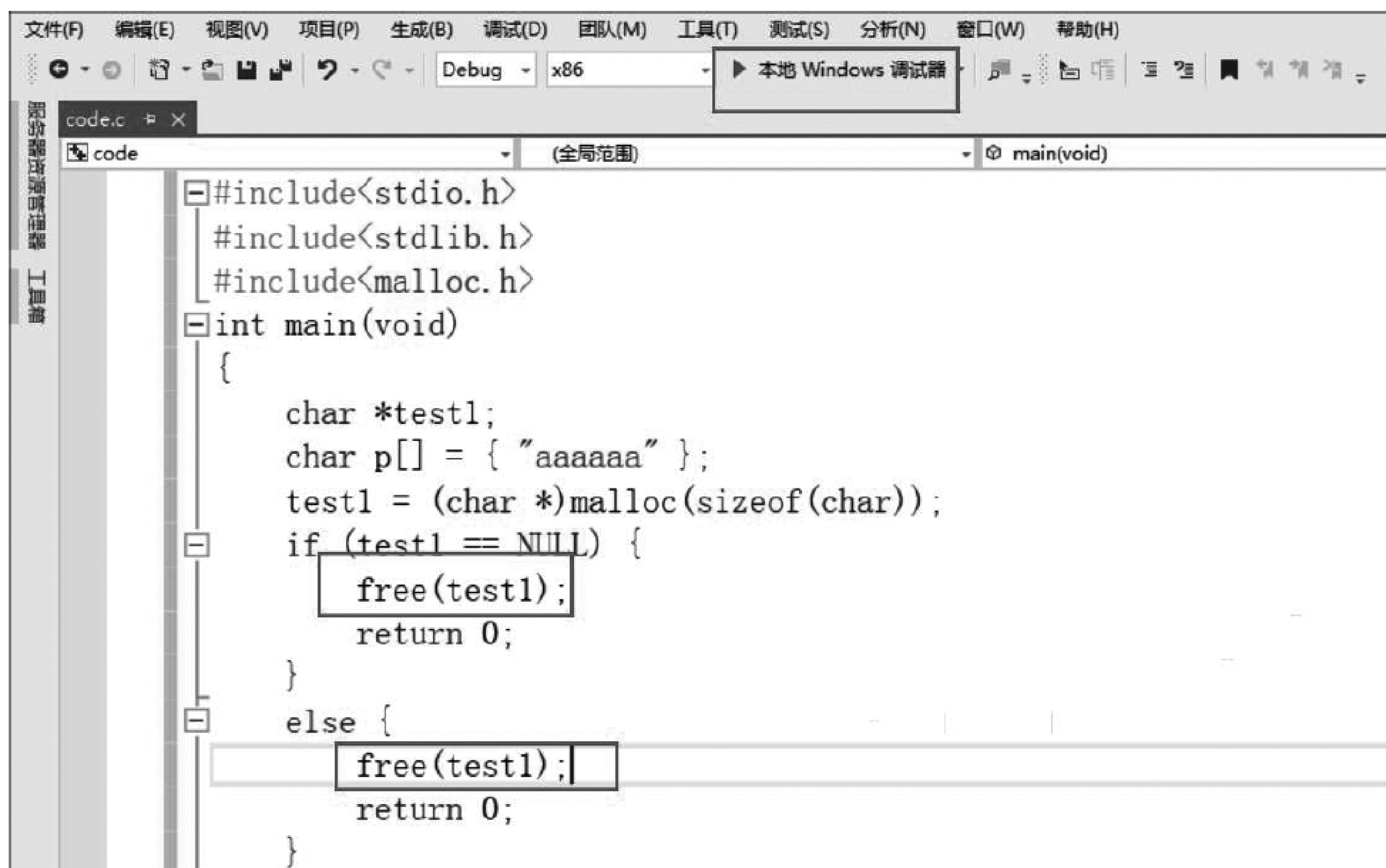


图 3-25 修改代码(3.1.5)

(8) 打开“C:\codemanager”文件夹，删除其中的所有文件然后关闭。

(9) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(10) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(11) 进入学生机“C:\ ”下的 codemanager，显示中间文件 codemanager.zip 生成成功。

(12) 进入 Chrome 浏览器代码卫士界面，选择“快速检测”→“＋发起快速检测”命令。

(13) “任务名称”输入“只释放动态分配的内存”，勾选“合规检测模板[C/C ++]”复选框。

(14) 选择“浏览”命令。

(15) 选择“本地磁盘(C:)”命令，选择右侧的 codemanager，单击“打开”按钮。

(16) 选择文件 codemanager，单击“打开”按钮。

(17) 返回“快速检测”界面，单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后，单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果，相关问题已经被修改，如图 3-26 所示。



图 3-26 查看结果(3.1.5)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考 free 一个静态分配的内存会造成什么错误。

3.1.6 代码控制流合规检测实验

【实验目的】

确保所编写的代码中控制流不会到达一个返回值非空的函数的末尾。

【知识点】

“控制流会到达一个返回值非空的函数的末尾”违规、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个任务调度模块，对传入的参数进行判断，在不同的情况下调用不同的库函数，获取该库函数的返回值并返回。需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：保证控制流不会到达一个返回值非空的函数的末尾。

【实验原理】

C 语言中的函数有其特定的返回值或者返回值为空，如果一个函数的返回值非空，但是控制流在特定情况下可以达到函数末尾并且没有返回值，那么调用该函数则会发生错误。因此在编写代码时应该避免这种错误。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-27 所示。

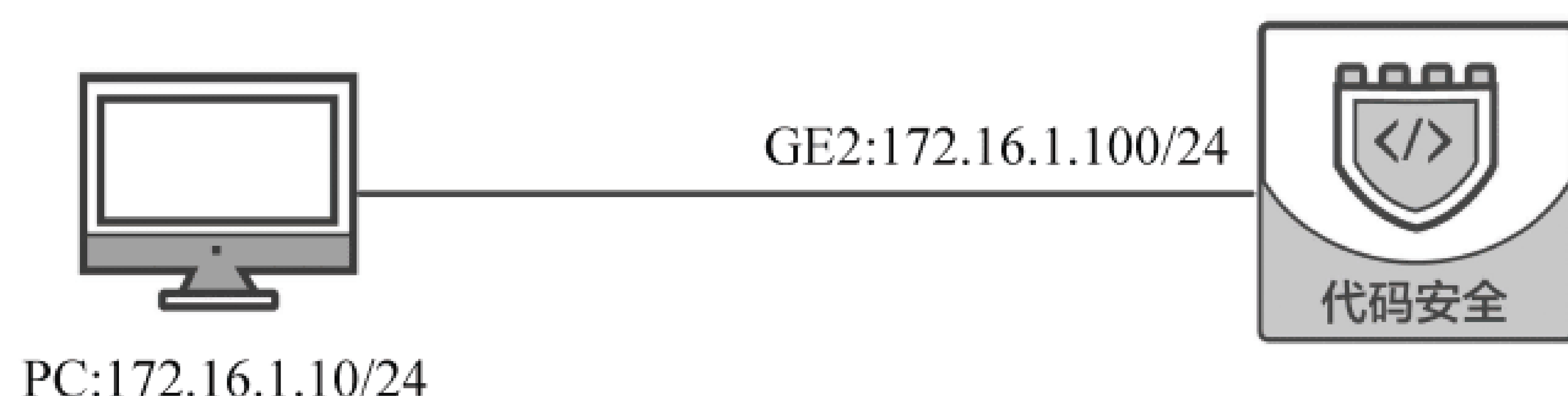


图 3-27　代码控制流合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在学生机桌面双击 Visual Studio 2015，显示主界面。

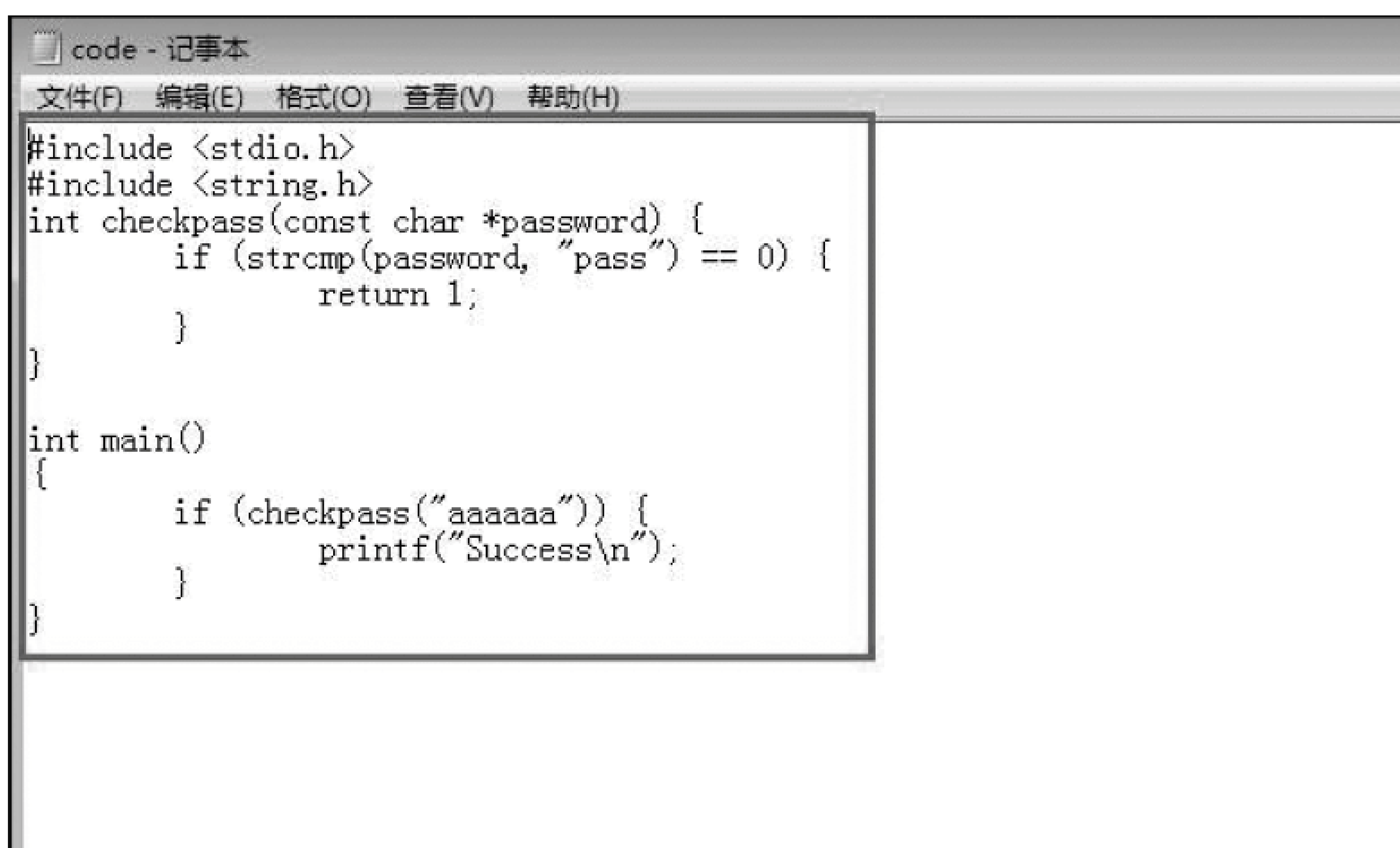
(3) 进入主界面,选择"文件"→"新建"→"项目"命令,新建一个项目。

(4) 进入"新建项目"界面,在左侧"模板"选择"Visual C++"命令,界面中间选项选择"空项目",下侧"名称"选项框内输入 code,单击"位置"右侧的"浏览"按钮,选择"C:\",其他默认选择。单击"确定"按钮。

(5) 再次进入 Visual Studio 2015 主界面,查看界面上"解决方案资源管理器"小窗口,已经新建好 code 项目。右击"源文件",在弹出的快捷菜单中选择"添加"→"新建项"命令,添加一个新项。

(6) 进入"添加新项"界面,左侧"已安装"默认选择"Visual C++"命令,在界面中间选择"C++文件(.cpp)"。在下侧"名称"选项框内输入 code.c,单击"添加"按钮。

(7) 打开"C:\"下的 code.txt,复制其中的代码,如图 3-28 所示。




```
#include <stdio.h>
#include <string.h>
int checkpass(const char *password) {
        if (strcmp(password, "pass") == 0) {
                return 1;
        }
}

int main()
{
        if (checkpass("aaaaaa")) {
                printf("Success\n");
        }
}
```

图 3-28　复制代码(3.1.6)

(8) 进入 Visual Studio 2015 代码编辑界面,在编辑框内粘贴代码,右击"解决方案资源管理器"窗口中的 code 项目。

(9) 选择"属性"命令。

(10) 选择"配置属性"中的"C/C++"下的"预处理器"选项。

(11) 选择"预处理器定义"命令。

(12) 单击右方的"下拉"按钮。

(13) 选择"编辑"命令。

(14) 在"预处理器定义"中输入"_CRT_SECURE_NO_WARNINGS"后单击"确定"按钮。

(15) 单击"确定"按钮。

(16) 单击"本地 Windows 调试器"按钮。

(17) 弹出"编译确认"对话框,单击"是"按钮。

(18) 选择"开始"→"VS 2015 开发人员命令提示"命令,打开命令行界面。

(19) 进入"管理员: VS 2015 开发人员命令提示"界面,在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(20) 进入学生机"C:\ "下的 codemanager 文件夹,显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器,网址输入"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(22) 在"高级"界面中单击"继续前往 172.16.1.100(不安全)"按钮。

(23) 界面跳转到代码卫士登录界面,"用户名"输入 admin,"密码"输入"admin123!@#",单击"登录"按钮。

(24) 选择"快速检测"→"+发起快速检测"命令。

(25) "任务名称"输入"保证控制流不会到达一个返回值非空的函数的末尾","开发语言"选中"C/C++"单选钮,勾选"合规检测模板[C/C++]"复选框。

(26) 选择"浏览"命令。

(27) 选择"本地磁盘(C:)"命令,选择右侧的 codemanager,单击"打开"按钮。

(28) 选择文件 codemanager,单击"打开"按钮。

(29) 返回"快速检测"界面,单击"发起检测"按钮。

【实验预期】

(1) 检测出代码中控制流会到达一个返回值非空的函数的末尾的违规。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1.查看检测结果,修改代码

(1) 在终端机打开 Chrome 浏览器,在地址栏中输入"https://172.16.1.100",进入代码安全保障系统登录界面。输入用户名 admin,密码为"admin123!@#",单击"登录"按钮。

(2) 选择"快速检测"命令。

(3) 选择"合规检测"命令,查看列表所示检测结果。

(4) 找到"保证控制流不会到达一个返回值非空的函数的末尾",单击右侧的"缺陷审计"按钮。

(5) 在左侧查看代码的缺陷,如图 3-29 所示。

(6) 打开"C:\code"文件夹,双击 code.sln,打开项目。

(7) 在代码编辑界面修改代码,单击"本地 Windows 调试器"按钮,如图 3-30 所示。

(8) 打开"C:\codemanager"文件夹,删除其中的所有文件然后关闭。

(9) 选择"开始"→"VS 2015 开发人员命令提示"命令,打开命令行界面。

(10) 进入"管理员: VS 2015 开发人员命令提示"界面,在命令行中输入命令"Manager -o C:\codemanager devenv C:\code\code.sln /rebuild"。

(11) 进入学生机"C:\ "下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(12) 进入 Chrome 浏览器代码卫士界面,选择"快速检测"→"+发起快速检测"命令。



图 3-29　“快速检测”界面(3.1.6)

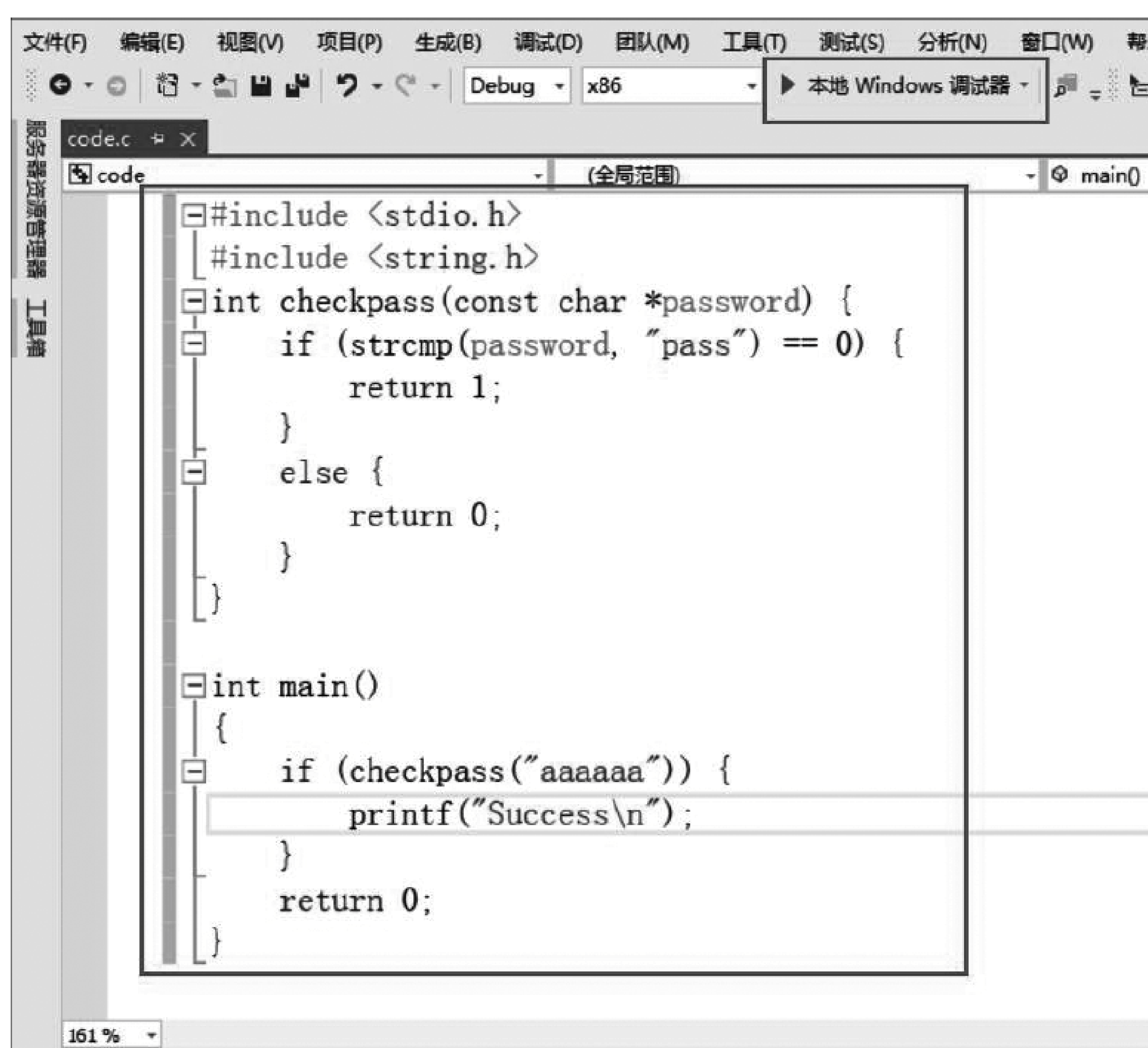


图 3-30　修改代码(3.1.6)

(13) “任务名称”输入“保证控制流不会到达一个返回值非空的函数的末尾”，“开发语言”选中“C/C++”单选钮，勾选“合规检测模板[C/C++]”复选框。

(14) 选择“浏览”命令。

(15) 选择“本地磁盘(C:)”命令，选择右侧的 codemanager，单击“打开”按钮。

(16) 选择文件 codemanager，单击“打开”按钮。

(17) 返回"快速检测"界面,单击"发起检测"按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后,单击任务右侧的"缺陷审计"按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 3-31 所示。



图 3-31 查看结果(3.1.6)

【实验思考】

(1) 请自行编写一个含有缺陷的代码并检测优化。

(2) 请思考当控制流到达一个返回值非空的函数的末尾时,该函数会返回什么。

3.1.7 字符串存储空间合规检测实验

【实验目的】

确保所编写的代码中不存在字符串存储越界的情况。

【知识点】

"存储越界"缺陷、C/C++合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个字符串处理模块,调用库函数对字符串进行处理,需要使用代码安全保障系统对编写的代码进行合规检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测:保证字符串的存储具有足够的空间容纳字符数据和 null 终结符。

【实验原理】

通过下面的例子来理解字符串存储越界错误的原理:

```
string p1 = "abcd";
char *p2 = "abcd";
char p3[] = "abcd";    //sizeof(p3) = 5;
```

字符串结尾都有系统自动补'\0',可用作判断字符串结束标志,字符数组和字符串最明显的区别就是字符串会被默认的加上结束符'\0'。

char p4[4]="abcd";//在定义时系统提示字符太长编译不通过。abcd 长度为 5,字符数组并不要求最后一个字符为'\0'.是否加入'\0',由程序员编码决定。但是我们写代码时要求字符数组初始化要求最后一个字符必须是'\0',类似 char p[4]={'a','b','c','d'};这样的定义是有风险的,虽然可以通过编译。如果字符串之后没有存储'\0',在使用字符串时会出现访问边界以外未知数据的情况,这样做是很危险的。

代码安全保障系统能够检测出代码中的缺陷,并给出详细描述以及修复建议,方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-32 所示。

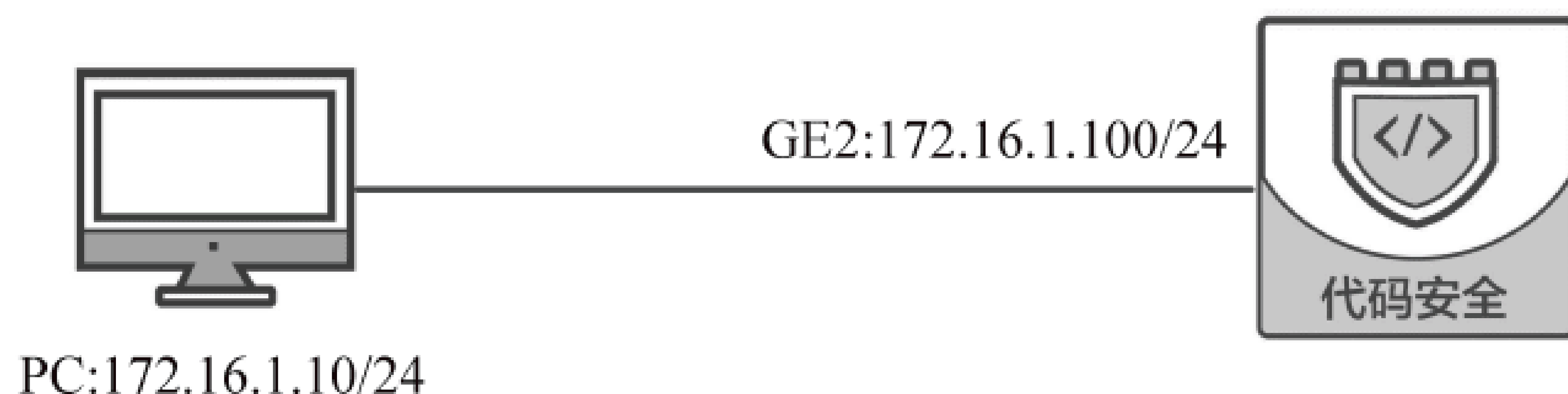


图 3-32　字符串存储空间合规检测实验拓扑图

【实验思路】

(1) 建立项目文件。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士,检测出缺陷。

(4) 依照检测结果对检测对象项目进行处理并再次检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 在学生机桌面双击 Visual Studio 2015,显示主界面。

(3) 进入主界面,选择"文件"→"新建"→"项目"命令,新建一个项目。

(4) 进入"新建项目"界面,在左侧"模板"选择 Visual C++命令,界面中间选项选择"空项目",下侧"名称"选项框内输入 code,单击"位置"右侧的"浏览"按钮,选择"C:\",其

他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.c，单击“添加”按钮。

(7) 打开“C:\”下的 code.txt，复制其中的代码，如图 3-33 所示。

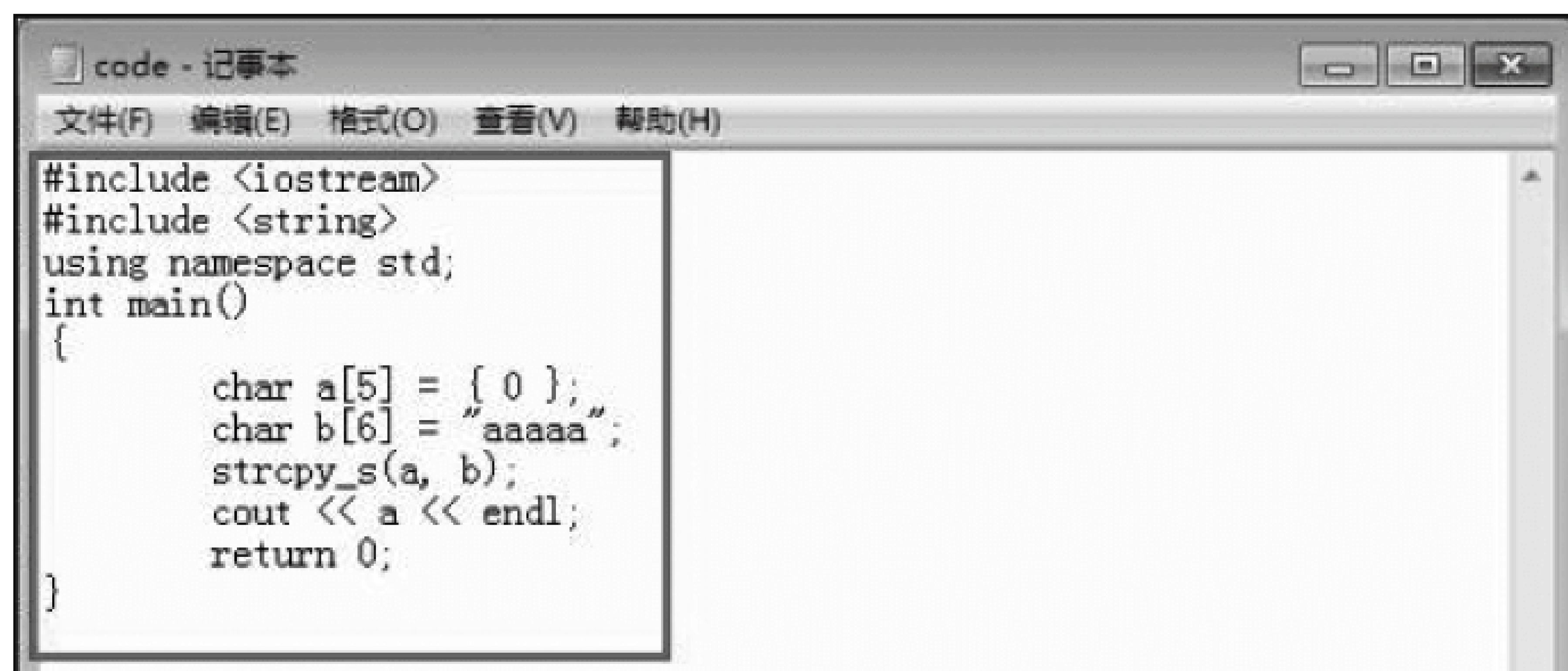


图 3-33　复制代码(3.1.7)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，右击“解决方案资源管理器”窗口中的 code 项目。

(9) 选择“属性”命令。

(10) 选择“配置属性”中的“C/C++”下的“预处理器”选项。

(11) 选择“预处理器定义”命令。

(12) 单击右方的“下拉”按钮。

(13) 选择“编辑”命令。

(14) 在“预处理器定义”中输入“_CRT_SECURE_NO_WARNINGS”后单击“确定”按钮。

(15) 单击“确定”按钮。

(16) 单击“本地 Windows 调试器”按钮。

(17) 弹出“编译确认”对话框，单击“是”按钮。

(18) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(19) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(20) 进入学生机“C:\ ”下的 codemanager 文件夹，显示中间文件 codemanager.zip 生成成功。

(21) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(22) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(23) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!

@＃”,单击“登录”按钮。

(24) 选择“快速检测”→“＋发起快速检测”命令。

(25) “任务名称”输入“保证字符串的存储具有足够的空间容纳字符数据和 null 终结符”,“开发语言”选中“C/C++”单选钮,勾选“合规检测模板[C/C++]”复选框。

(26) 选择“浏览”命令。

(27) 单击“本地磁盘(C:)”按钮,选择右侧的 codemanager,单击“打开”按钮。

(28) 选择文件 codemanager,单击“打开”按钮。

(29) 返回“快速检测”界面,单击“发起检测”按钮。

【实验预期】

(1) 检测出代码中释放后使用的缺陷。

(2) 给出该缺陷的详细描述和修复建议。

【实验结果】

1. 检测出代码中释放后使用的缺陷

(1) 主机终端打开 Chrome 浏览器,在地址栏中输入“https://172.16.1.100”,进入代码安全保障系统登录界面。输入用户名 admin,密码为“admin123!@＃”,单击“登录”按钮。

(2) 选择“快速检测”命令。

(3) 找到“保证字符串的存储具有足够的空间容纳字符数据和 null 终结符”,单击右侧的“缺陷审计”按钮。

(4) 在左侧查看代码的缺陷,如图 3-34 所示。



图 3-34　“快速检测”界面(3.1.7)

(5) 打开“C:\code”文件夹,双击 code.sln,打开项目。

(6) 在代码编辑界面修改代码,单击“本地 Windows 调试器”按钮,如图 3-35 所示。

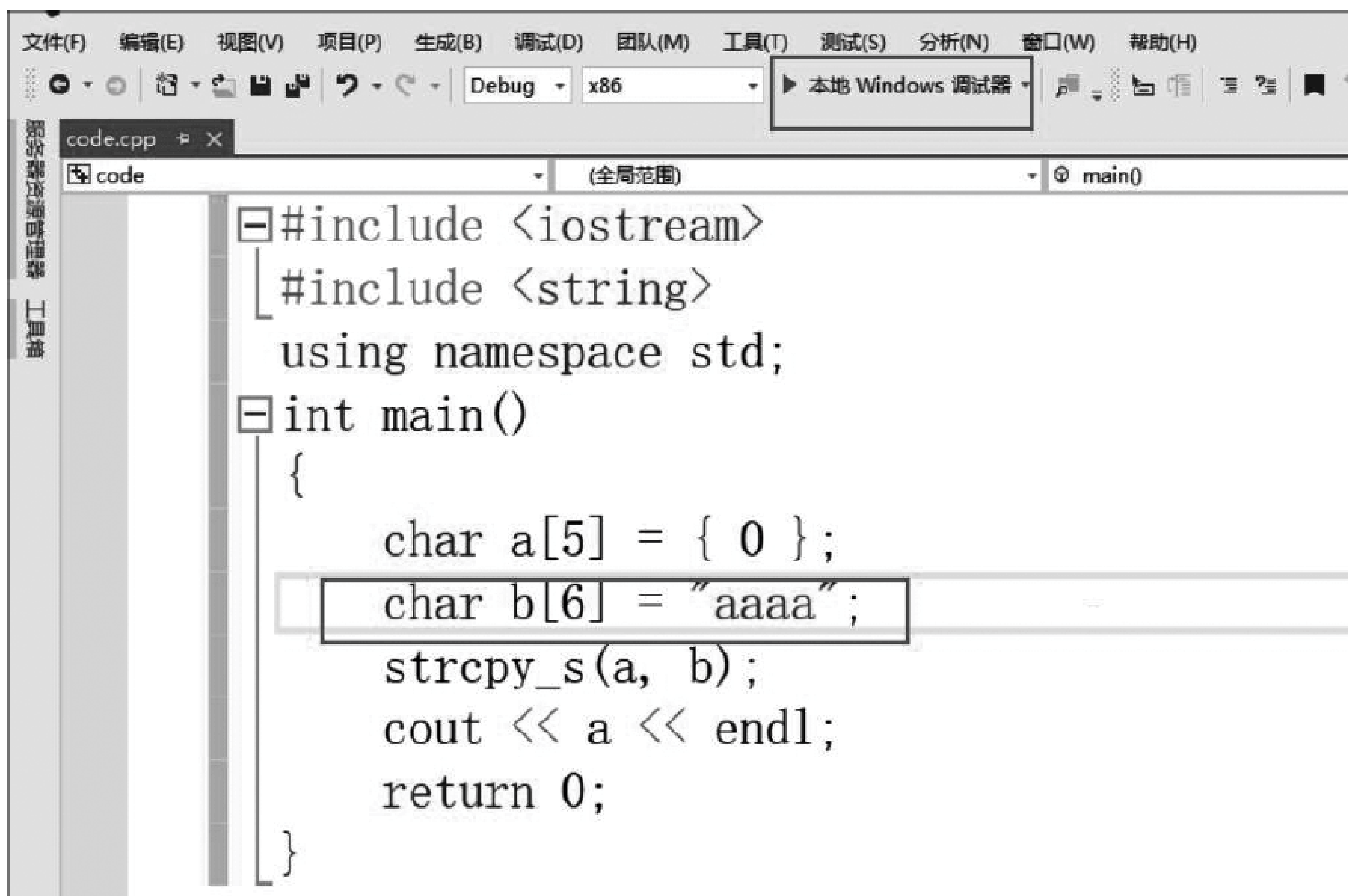


图 3-35　修改代码(3.1.7)

(7) 打开“C:\codemanager”文件夹,删除其中的所有文件然后关闭。

(8) 选择“开始”→“VS 2015 开发人员命令提示”命令,打开命令行界面。

(9) 进入“管理员: VS 2015 开发人员命令提示”界面,在命令行中输入命令“Manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(10) 进入学生机“C:\ ”下的 codemanager,显示中间文件 codemanager.zip 生成成功。

(11) 进入 Chrome 浏览器代码卫士界面,选择“快速检测”→“+发起快速检测”命令。

(12) “任务名称”输入“保证字符串的存储具有足够的空间容纳字符数据和 null 终结符”,“开发语言”选中“C/C++”单选钮,勾选“合规检测模板[C/C++]”复选框。

(13) 选择“浏览”命令。

(14) 选择“本地磁盘(C:)”命令,选择右侧的 codemanager,单击“打开”按钮。

(15) 选择文件 codemanager,单击“打开”按钮。

(16) 返回“快速检测”界面,单击“发起检测”按钮。

2. 对比修改前后检测结果

(1) 等待任务检测完成后,单击任务右侧的“缺陷审计”按钮。

(2) 在左侧查看检测结果,相关问题已经被修改,如图 3-36 所示。

【实验思考】

请自行编写一个含有缺陷的代码并检测优化。



图 3-36　查看结果(3.1.7)

3.2　Java 合规检测

3.2.1　代码重用 Java 标准库已经公开的标识符合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“重用 Java 标准库已经公开的标识符”的缺陷，并对发现的“重用 Java 标准库已经公开的标识符”的缺陷进行针对性的修复，确保所编写的代码中不存在“重用 Java 标准库已经公开的标识符”的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个公共类库代码模块，定义了一些与 JDK API 类名相同的类，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要重用 Java 标准库已经公开的标识符。

【实验原理】

当一个程序员使用和公开类相同的名字时，例如 Vector，对后来的维护者来说，他可能不知道这个标识并不是指 Java. util. Vector，并且可能会无意地使用这个自定义的 Vector 类而不是原有的 Java.util.Vector 类，从而导致不可预计的程序行为。因此不要重用那些在 Java 标准库中已经使用过的公共的标识，公共的工具类、接口或者包。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-37 所示。

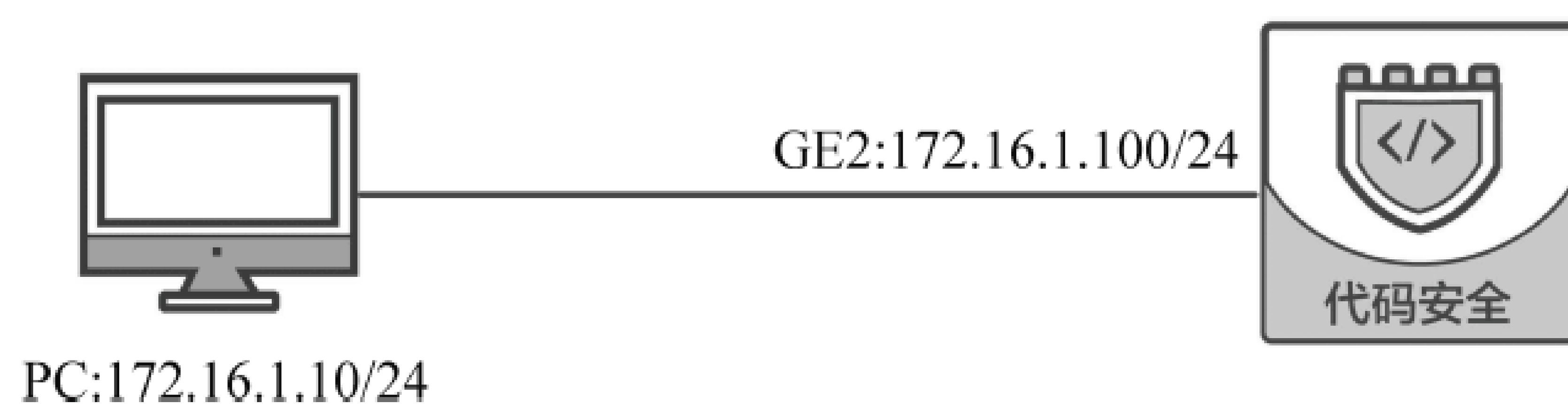


图 3-37　代码重用 Java 标准库已经公开的标识符合规检测实验拓扑图

【实验思路】

（1）发起 Java 代码合规检测。

（2）根据修改建议对代码进行修改。

（3）检测修改效果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开“C:\code\code\src\code”目录下的 code.java 文件。

（3）程序中部分代码展示如图 3-38 所示。

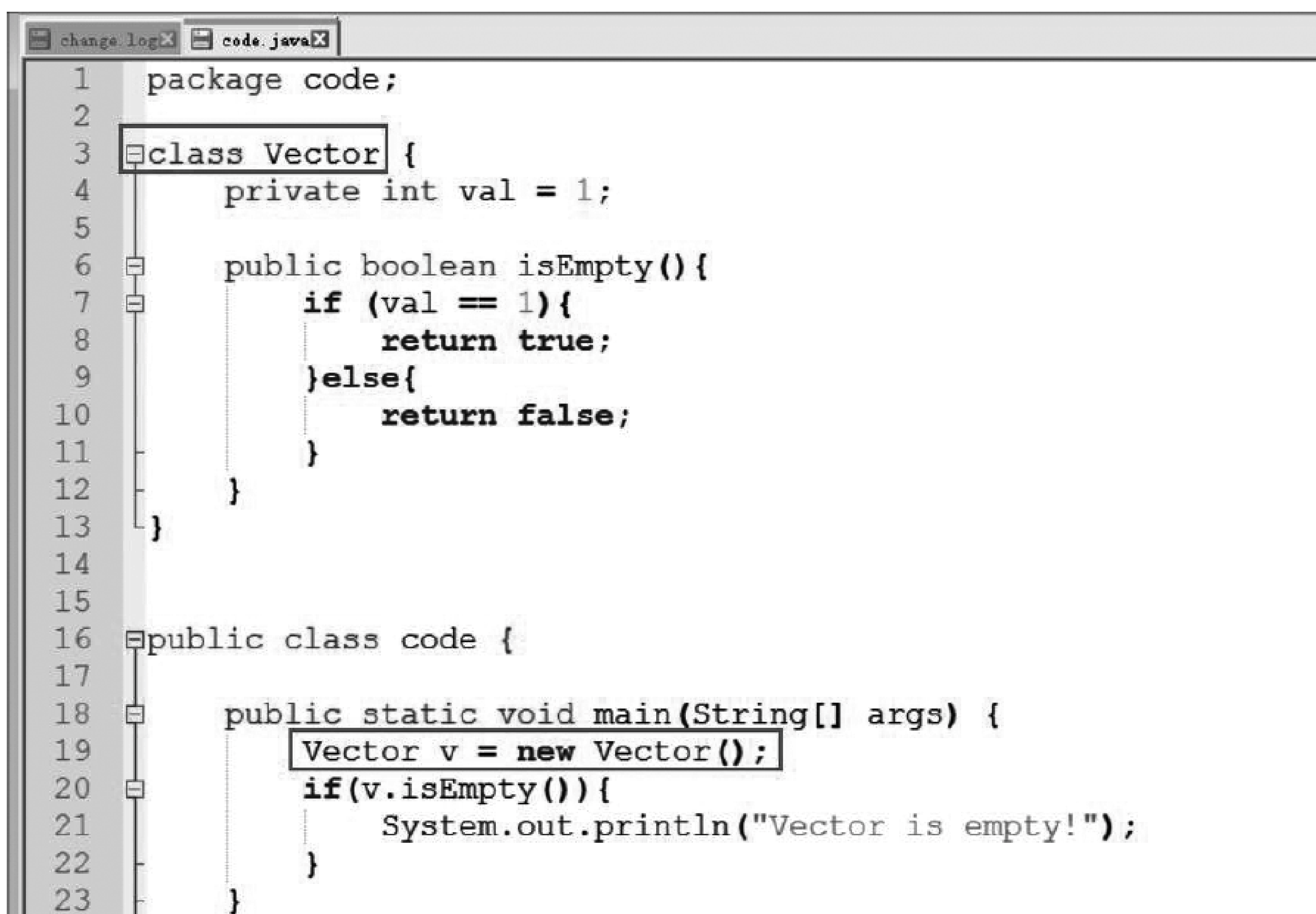


图 3-38　部分代码展示(3.2.1)

（4）打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

（5）单击“继续前往 172.16.1.100”按钮。

（6）跳转到登录界面，输入用户名与密码（默认用户名为 admin，密码为“admin123!@#”），单击“登录”按钮。

（7）进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

（8）在配置信息界面中，“任务名称”输入“不要重用 Java 标准库已经公开的标识符”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选框，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

（9）发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧的“缺陷审计”按钮。

（10）可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“中”。单击左侧框中的“＋”按钮，展开目录，如图 3-39 所示。



图 3-39　展开界面(3.2.1)

（11）选择“code.java(3)”命令，快速定位有问题的代码。

（12）选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。

（13）选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后对应的缺陷消失。

【实验结果】

（1）最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-40 所示。

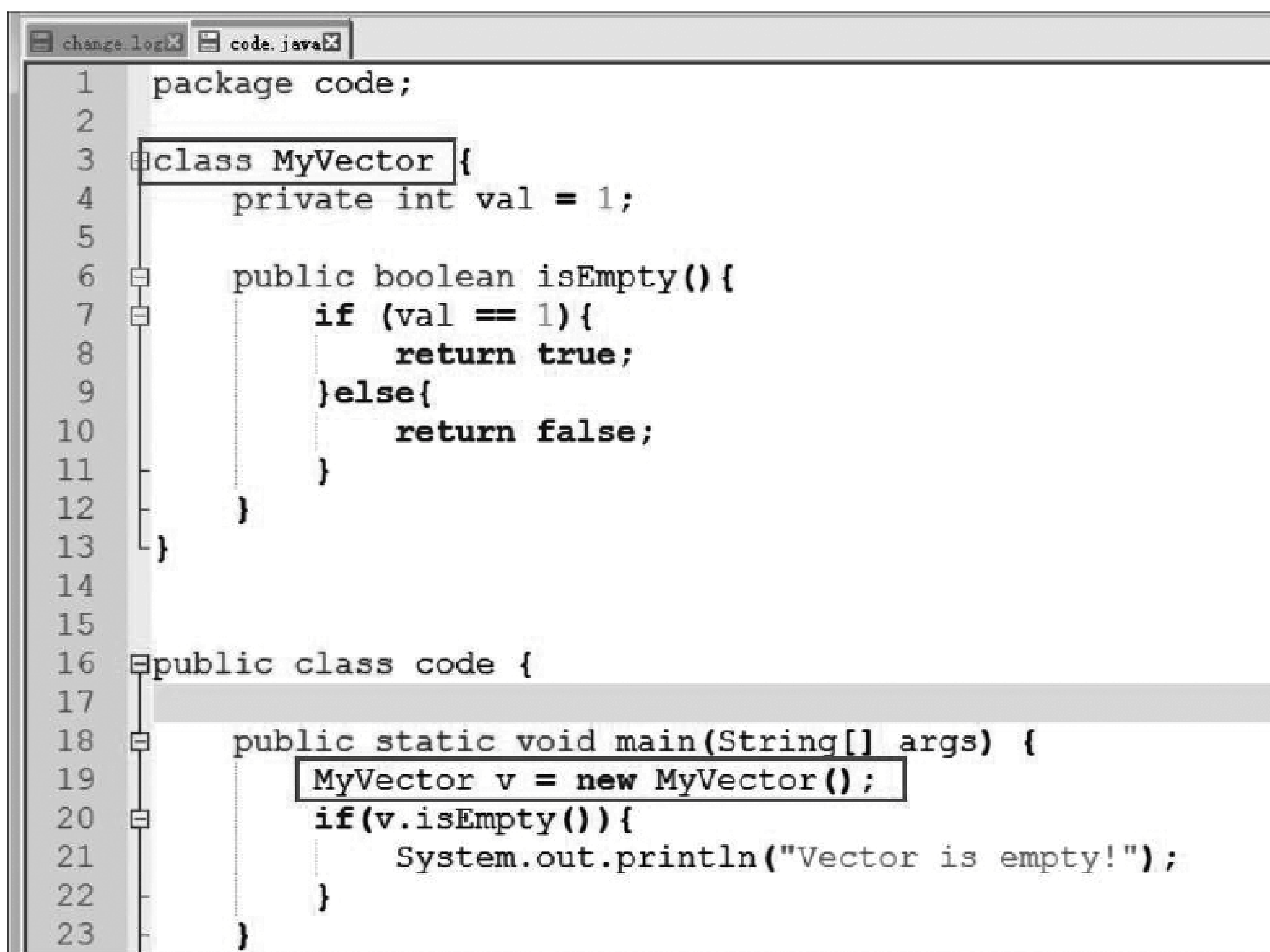

```
1   package code;
2
3   class MyVector {
4       private int val = 1;
5
6       public boolean isEmpty(){
7           if (val == 1){
8               return true;
9           }else{
10              return false;
11          }
12      }
13  }
14
15
16  public class code {
17
18      public static void main(String[] args) {
19          MyVector v = new MyVector();
20          if(v.isEmpty()){
21              System.out.println("Vector is empty!");
22          }
23      }
```

图 3-40　代码修改(3.2.1)

(2) 右击 code 文件夹,在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”,单击“立即压缩”按钮。

(4) 打开浏览器,在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“不要使用 Java 标准库已经公开的标识符修复检测”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code-fix.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“缺陷总数”减少了,单击最右侧的“缺陷审计”按钮(图中最右框位置),如图 3-41 所示。

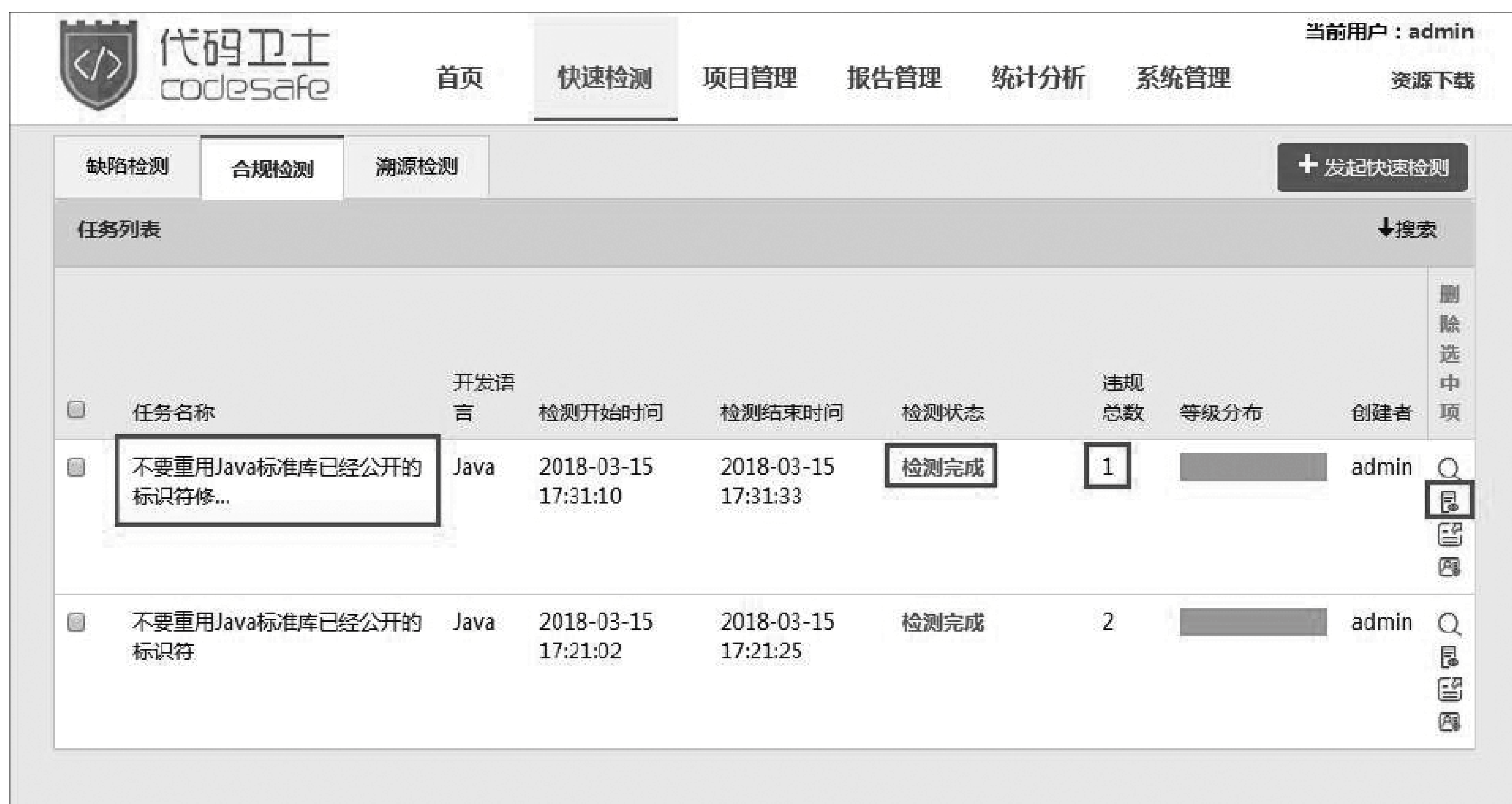


图 3-41　检测完成(3.2.1)

(7) 可以看到缺陷已经修复，如图 3-42 所示。



图 3-42　缺陷修复(3.2.1)

【实验思考】

(1) Java 中是如何定义标识符的？

(2) Java 中的关键字可以作为标识符吗？

3.2.2　代码使用 Object.equals( ) 方法来比较两个数组合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“使用 Object.equals() 方法来比较两个数组”的缺陷，并对发现的“使用 Object.equals() 方法来比较两个数组”的缺陷进行针对性的修复，确保所编写的代码中不存在“使用 Object.equals() 方法来比较两个数组”的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个 bean 对象比较代码模块，使用了 equals() 方法比较两个数组，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要使用 Object.equals() 方法来比较两个数组。

【实验原理】

数组不会重写 Object.equals()方法，而 equals()方法比较的是数组引用而不是数组的内容值。程序必须使用两个参数的 Arrays.equals()方法来比较两个数组的内容。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-43 所示。

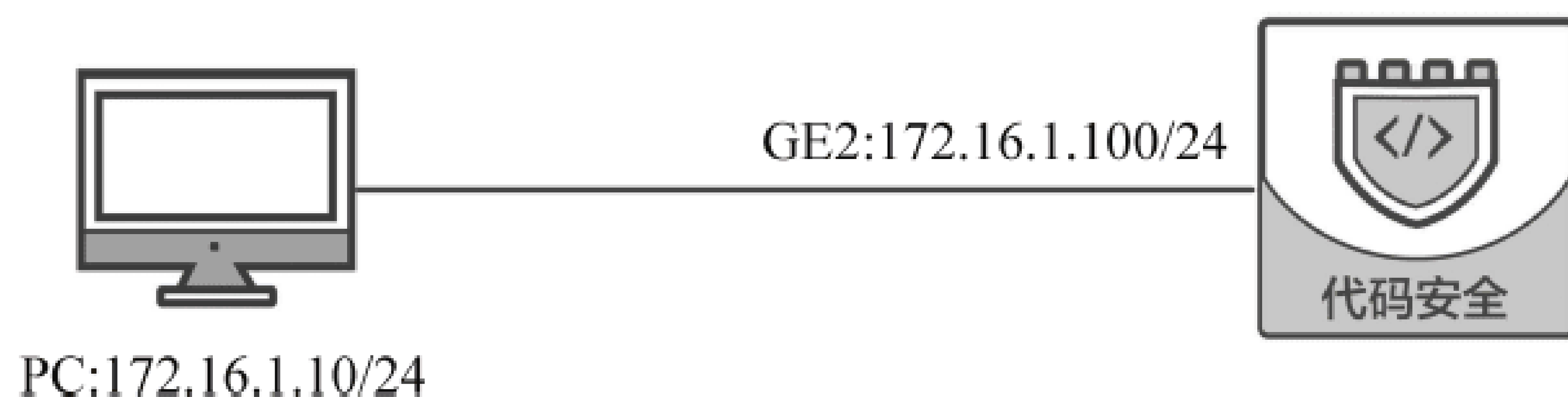


图 3-43 代码使用 Object.equals()方法来比较两个数组合规检测实验拓扑图

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-44 所示。

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要使用 Object.equals 方法来比较两个数组”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧

```
16
17      public void setGrades(double[] grades) {
18          this.grades = grades;
19      }
20  }
21
22  public class code {
23
24      public static void main(String[] args) {
25          student s1 = new student();
26          student s2 = new student();
27          System.out.println(arrayEquals(s1.getGrades(),s2.getGrades()))
28      }
29
30      public static boolean arrayEquals(double[] a,double[] b){
31          return a.equals(b);
32      }
33
34  }
35
```

图 3-44　部分代码展示(3.2.2)

的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷,并判断该缺陷的风险等级为“中”。单击左侧框中的“+”展开目录,如图 3-45 所示。



图 3-45　展开界面(3.2.2)

(11) 选择“code.java(31)”命令,快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后,对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-46 所示。

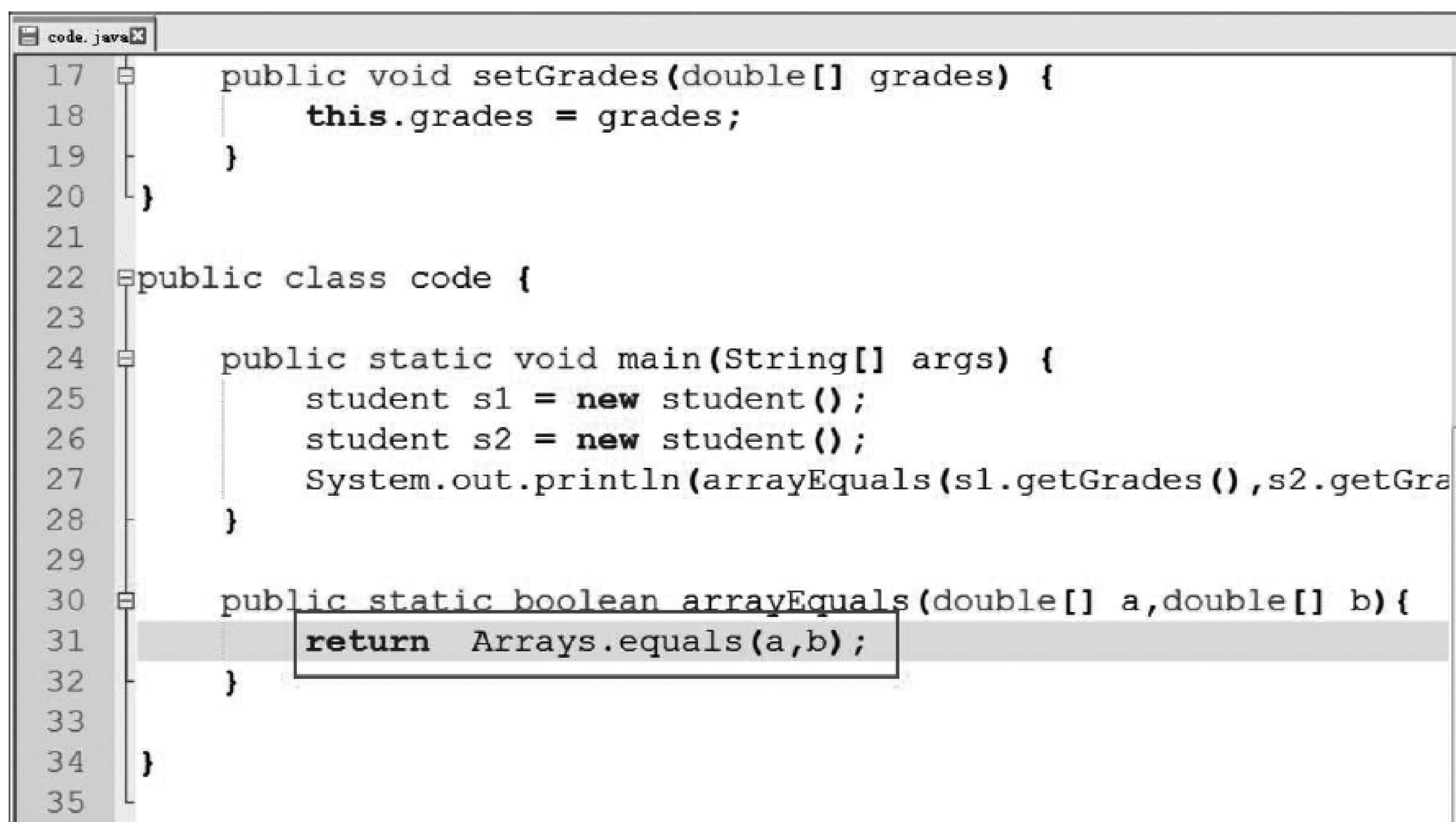


图 3-46　代码修改(3.2.2)

(2) 右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“不要使用 Java 标准库已经公开的标识符修复检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮(图中最右框位置)，如图 3-47 所示。



图 3-47　检测完成(3.2.2)

(7) 可以看到缺陷已经修复,如图3-48所示。



图3-48 缺陷修复(3.2.2)

【实验思考】

(1) Java中比较两个数组时,使用运算符"=="和方法Arrays.equals()的区别是什么?

(2) Java中什么情况会使用运算符"=="比较对象?

3.2.3 代码使用相等操作符比较封装的基础数据类型合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在"使用相等操作符比较封装的基础数据类型"的缺陷,并对发现的"使用相等操作符比较封装的基础数据类型"的缺陷进行针对性的修复,确保所编写的代码中不存在"使用相等操作符比较封装的基础数据类型"的缺陷。

【知识点】

Java代码合规检测。

【场景描述】

A公司研发部工程师小王编写了一个bean对象比较代码模块,使用了相等操作符比较基础类型的封装类,需要使用代码安全保障系统对编写的代码进行合规检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测:不要使用相等操作符比较封装的基础数据类型。

【实验原理】

两个封装的基础数据类型的值是不能直接使用“==”或“! =”操作符来比较的，因为这些操作符比较的是对象引用而不是对象的值。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-49 所示。

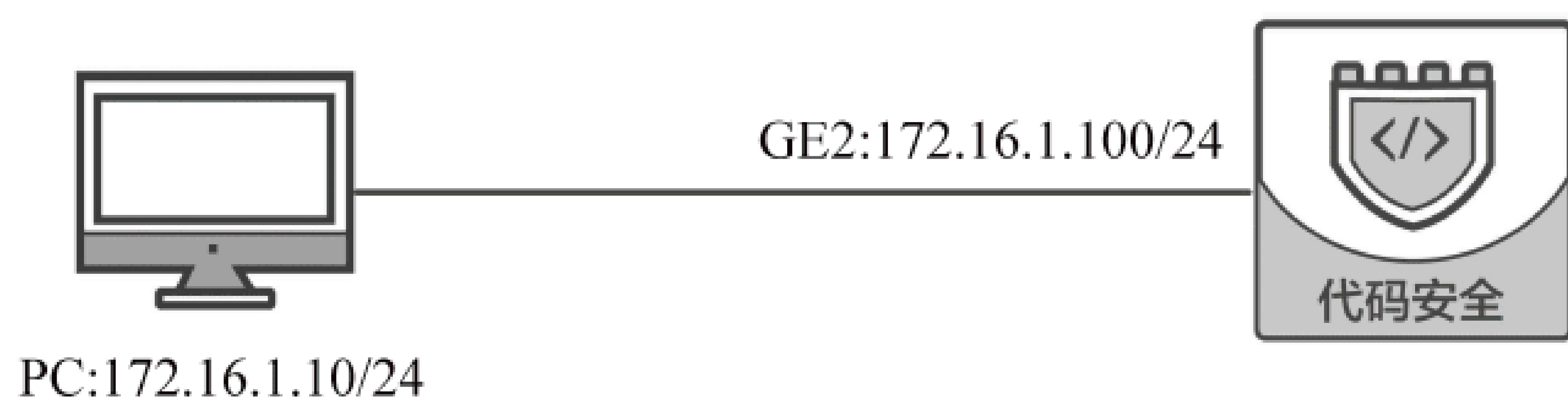


图 3-49　代码使用相等操作符比较封装的基础数据类型合规检测实验拓扑图

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-50 所示。

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要使用相等操作符比较封装的基础数据类型”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧

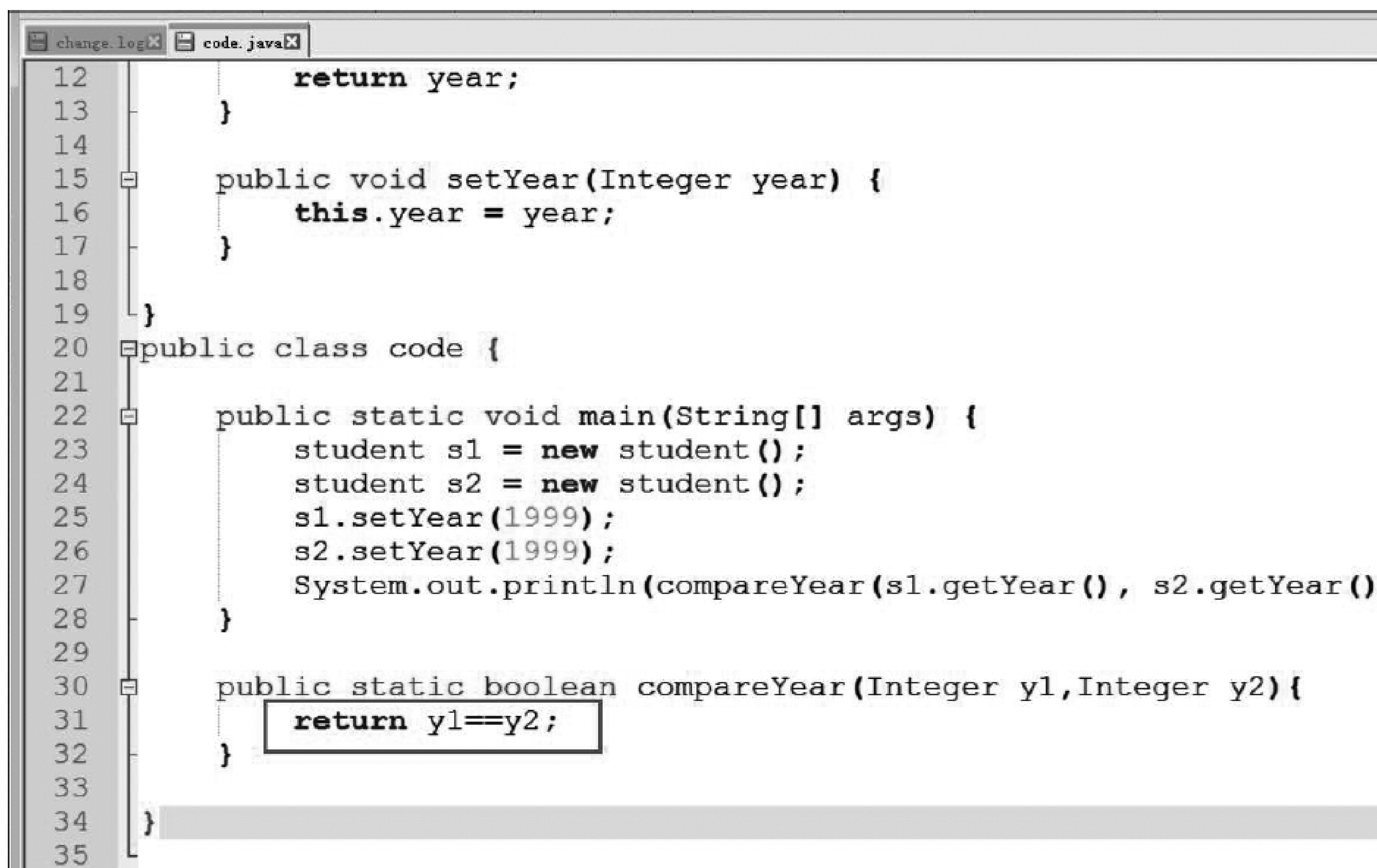


图 3-50　部分代码展示(3.2.3)

的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷,并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录,如图 3-51 所示。



图 3-51　展开界面(3.2.3)

(11) 选择 code.java 命令,快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后，对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-52 所示。

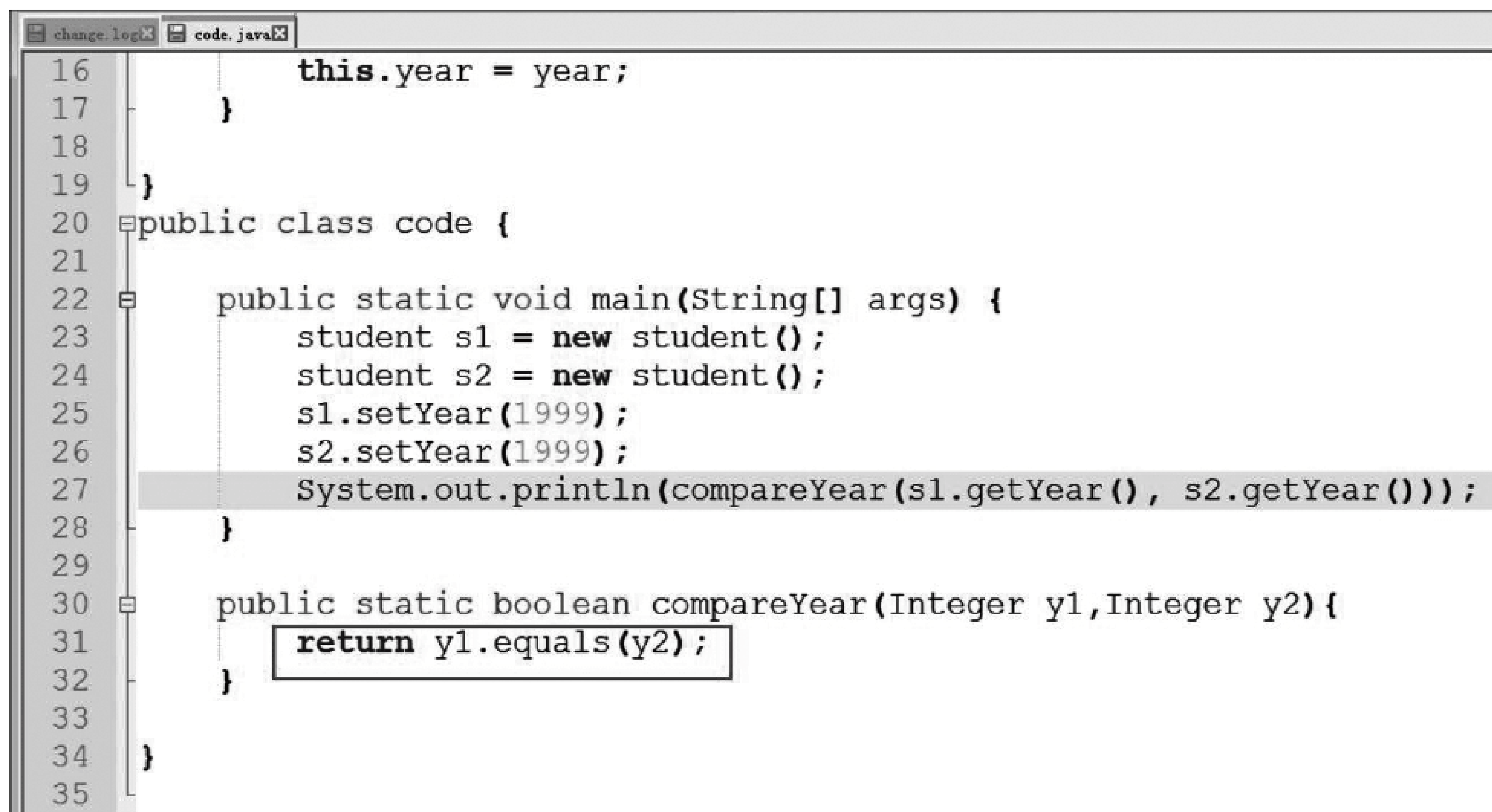


图 3-52 代码修改(3.2.3)

(2) 右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“不要使用相等操作符比较封装的基础数据类型”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮。

(7) 可以看到缺陷已经修复，如图 3-53 所示。

【实验思考】

(1) 为什么不能用相等操作符来比较封装的基础数据类型?

(2) 可以使用相等操作符比较未封装的基础数据类型吗?

3.2.4 代码使用浮点数变量作为循环计数器合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“使用浮点数变量作为



图 3-53　缺陷修复(3.2.3)

循环计数器”的缺陷，并对发现的“使用浮点数变量作为循环计数器”的缺陷进行针对性的修复，确保所编写的代码中不存在“使用浮点数变量作为循环计数器”的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个订单金额计算代码模块，使用了浮点数变量作为循环计数器，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要使用浮点数变量作为循环计数器。

【实验原理】

有限精度 IEEE 754 浮点类型不能表达：精确的分数、精确的小数和大数值的数字。例如，0.1f 会被处理成可以用 float 类型表达的最接近的数值，所以在每次循环中，增加的数值会是一个略大于 0.1 的值，不能产生预期的结果。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-54 所示。

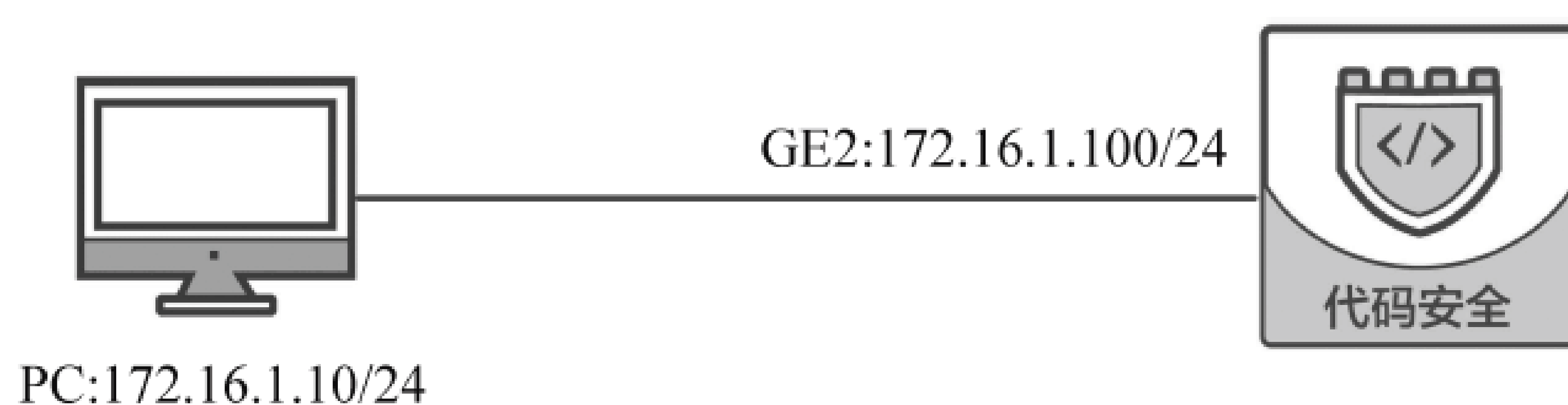


图 3-54　代码使用浮点数变量作为循环计数器合规检测实验拓扑图

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-55 所示。

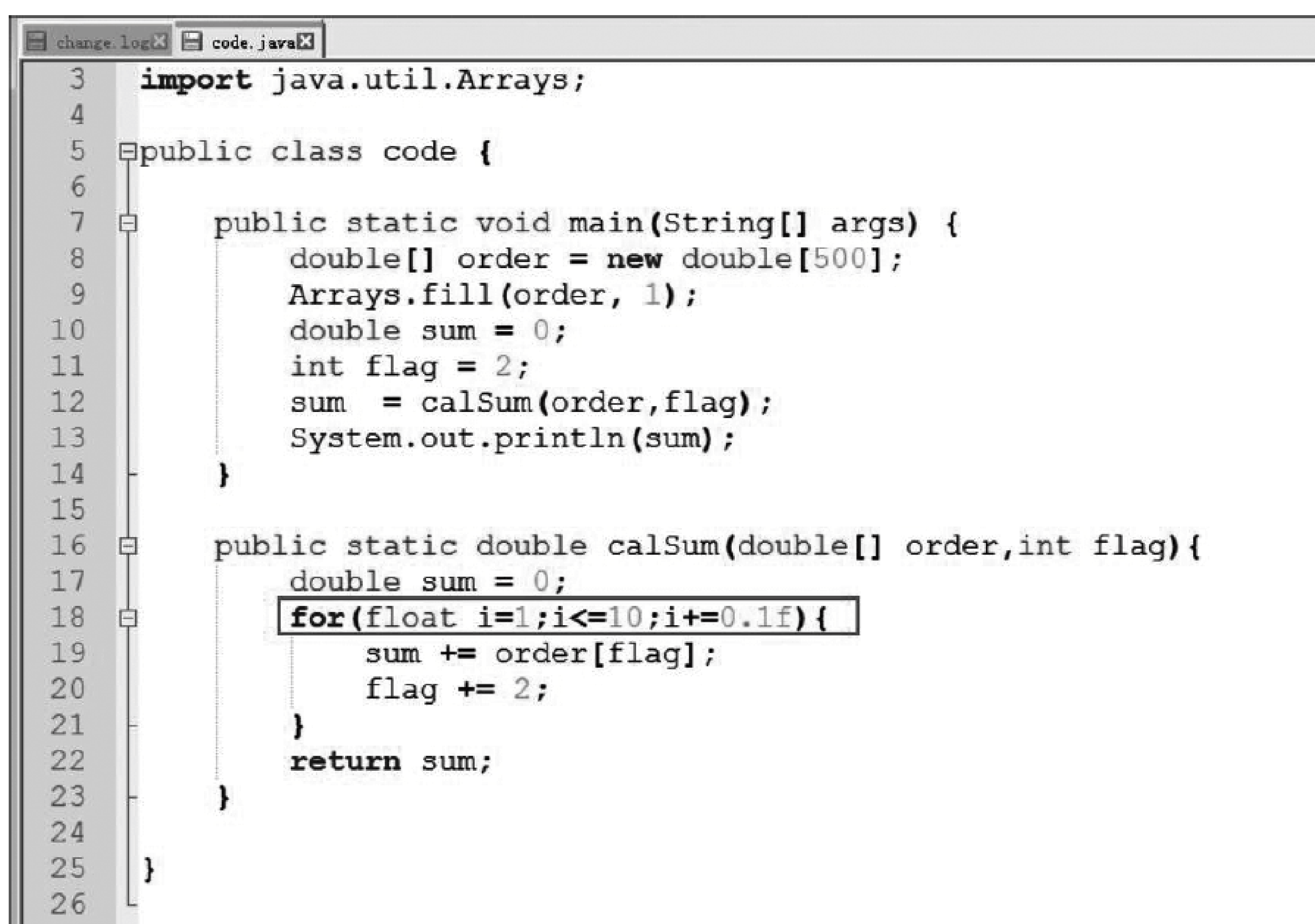


图 3-55　部分代码展示(3.2.4)

(4) 打开 Chrome 浏览器,输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要使用浮点数变量作为循环计数器”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录，如图 3-56 所示。



图 3-56　展开界面(3.2.4)

(11) 选择 code.java 命令，快速定位有问题的代码。

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后，对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-57 所示。

(2) 右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中，“任务名称”输入“不要使用浮点数变量作为循环计数器”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击 “上传文件”右侧的

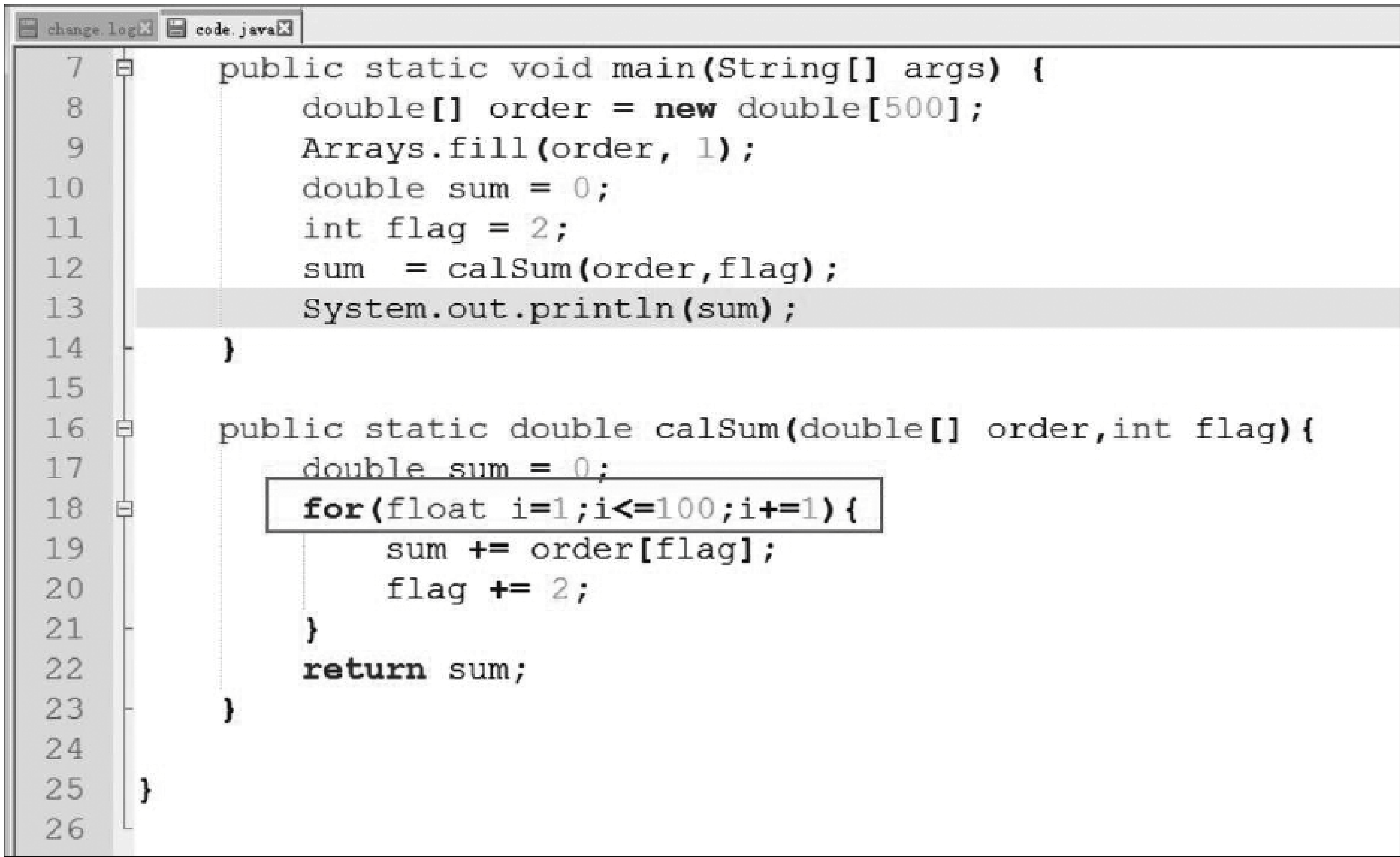


图 3-57　代码修改(3.2.4)

“浏览”按钮,选择“C:\code”目录下的“code-fix.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“缺陷总数”减少了,单击最右侧的“缺陷审计”按钮。

(7) 可以看到缺陷已经修复,如图 3-58 所示。



图 3-58　缺陷修复(3.2.4)

【实验思考】

(1) 描述 IEEE 754 浮点数的表示范围。

(2) IEEE 754 浮点数不能精确表示哪些数字?

3.2.5 代码捕获 NullPointerException 或者任何它的基类合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在"捕获 NullPointerException 或者任何它的基类"的缺陷,并对发现的"捕获 NullPointerException 或者任何它的基类"的缺陷进行针对性的修复,确保所编写的代码中不存在"捕获 NullPointerException 或者任何它的基类"的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个公共异常处理代码模块,捕获了 NullPointerException 异常,需要使用代码安全保障系统对编写的代码进行合规检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块合规检测:不要捕获 NullPointerException 或者任何它的基类。

【实验原理】

程序不能捕捉 java.lang.NullPointerException。首先,捕捉 NullPointerException 而不是进行简单的空引用检测,会在性能上付出相当大的代价。其次,当 try 程序段中多个表达式都有可能抛出 NullPointerException 时,判断抛出异常的表达式是困难的甚至是不可能的,这是因为 catch 程序段会去处理 try 程序段中任何位置抛出的 NullPointerException。

代码安全保障系统能够检测出代码中的缺陷,并给出详细描述以及修复建议,方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-59 所示。

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

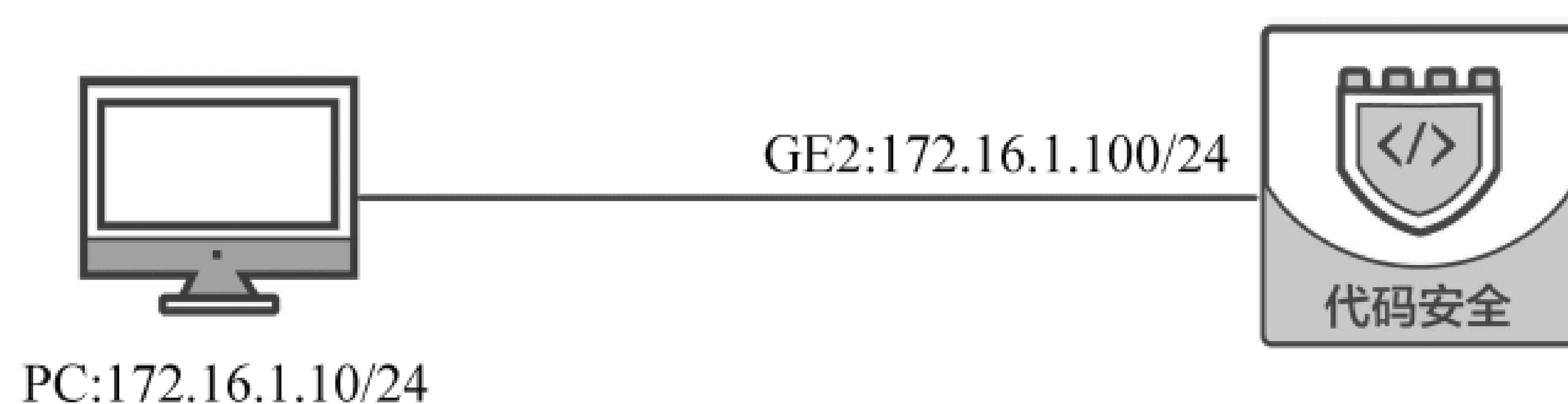


图 3-59　代码捕获 NullPointerException 或者任何它的基类合规检测实验拓扑图

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-60 所示。

```
1   package code;
2
3   import java.util.Locale;
4
5   public class code {
6
7       public static void main(String[] args) {
8           String name = " Capne";
9           System.out.println(isName(name));
10      }
11
12      public static boolean isName(String s){
13          try{
14              String names[] = s.split(" ");
15              if(names.length != 2)
16                  return false;
17              return (isCapitalized(names[0])) && (isCapitalized(names[1]));
18          }catch(NullPointerException e){
19              return false;
20          }
21      }
22
23      public static boolean isCapitalized(String s){
24          if(s == null){
```

图 3-60　部分代码展示(3.2.5)

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要捕获 NullPointerException 或者任何它的基类”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧

的“缺陷审计”按钮。

（10）可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“低”。单击左侧框中的“＋”按钮展开目录，如图 3-61 所示。



图 3-61　展开界面(3.2.5)

（11）选择 code.java 命令，快速定位有问题的代码。

（12）选择“详细信息”命令可以查看问题代码中该条缺陷的具体信息。

（13）选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后对应的缺陷消失。

【实验结果】

（1）最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-62 所示。

（2）右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

（3）“压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

（4）打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

（5）在配置信息界面中，“任务名称”输入“不要捕获 NullPointerException 或者任何它的基类”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

（6）在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮。

（7）可以看到缺陷已经修复，如图 3-63 所示。

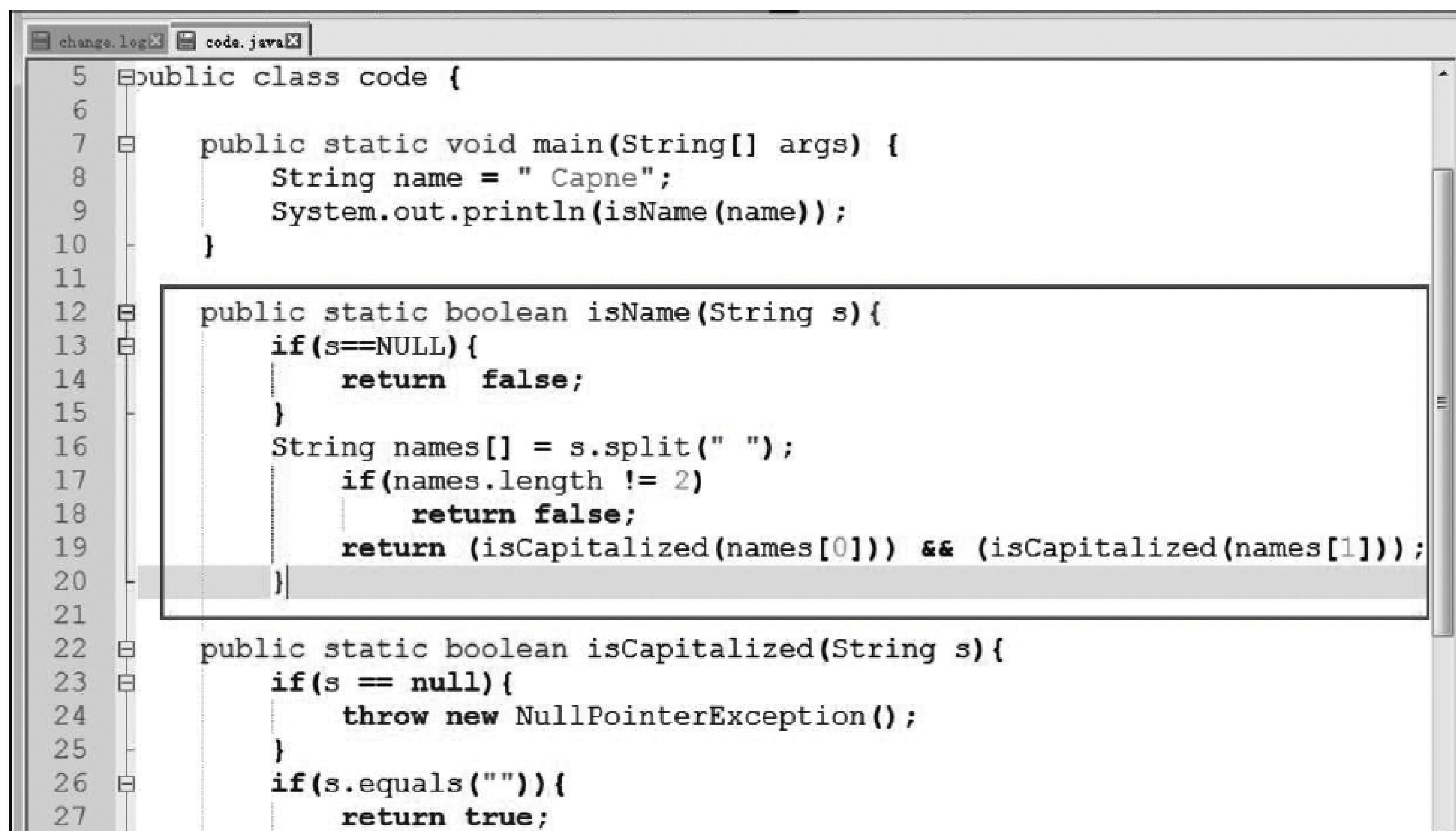


图 3-62 代码修改(3.2.5)



图 3-63 缺陷修复(3.2.5)

【实验思考】

(1) Java 中造成空指针异常的原因有哪些?

(2) Java 中如何避免出现空指针异常?

3.2.6 代码实例锁的使用合规检测实验

【实验目的】

通过使用代码安全保障系统检测上传代码模块是否存在“使用实例锁来保护共享静

态数据”的缺陷，并对发现的“使用实例锁来保护共享静态数据”的缺陷进行针对性的修复，确保所编写的代码中不存在“使用实例锁来保护共享静态数据”的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个多线程代码模块，使用了实例锁来保护共享静态数据，需要使用代码安全保障系统对编写的代码进行合规检测。

通过代码安全保障系统的检测结果调整和优化代码模块，请帮助小王完成代码模块的合规检测：不要使用实例锁来保护共享静态数据。

【实验原理】

程序不应使用实例锁来保护静态共享数据，原因在于实例锁在两个或者多个实例的情况下是无效的。因此，如果不使用一个静态的锁对象，会导致共享状态在并发进入时是不被保护的。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-64 所示。

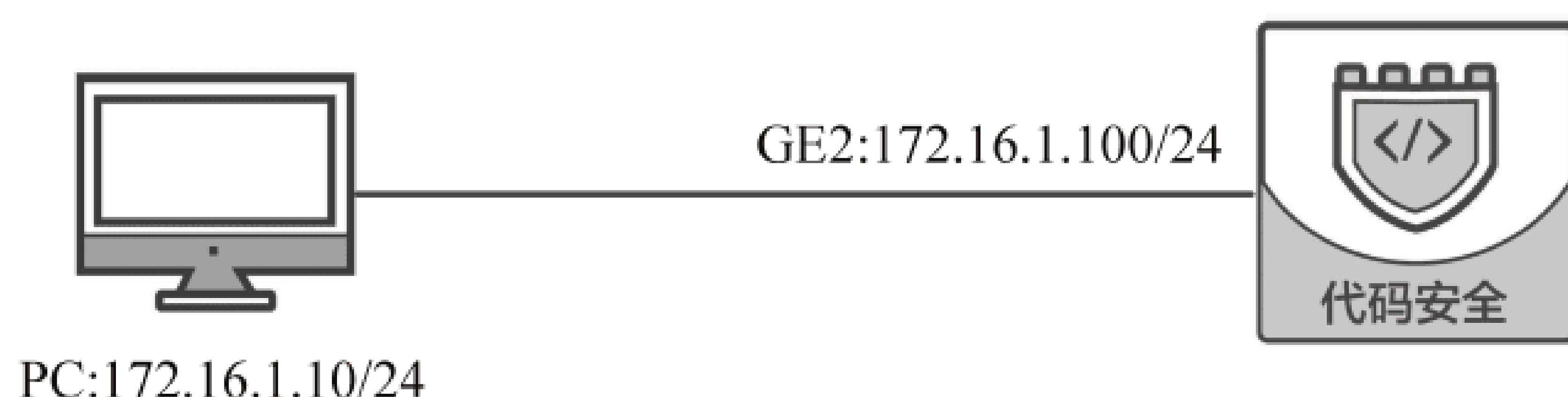


图 3-64　代码实例锁的使用合规检测实验拓扑图

【实验思路】

（1）发起 Java 代码合规检测。

（2）根据修改建议对代码进行修改。

（3）检测修改效果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开“C:\code\code\src\code”目录下的 code.java 文件。

（3）程序中部分代码展示如图 3-65 所示。

```
1  package code;
2
3  public class code implements Runnable{
4
5      private static volatile int counter;
6      //...
7      private final Object lock = new Object();
8
9      @Override
10     public void run() {
11         synchronized (lock) {
12             counter++;
13             //...
14         }
15     }
16
17     public static void main(String [] args) {
18         for(int i=0;i<2;i++) {
19             new Thread(new code()).start();
20         }
21     }
22 }
```

图 3-65 部分代码展示(3.2.6)

(4) 打开 Chrome 浏览器,输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(7) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中,“任务名称”输入“不要使用实例锁来保护共享静态数据”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。当检测状态为“检测完成”时,单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷,并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录,如图 3-66 所示。

(11) 选择“code.java(31)”命令,快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。

【实验预期】

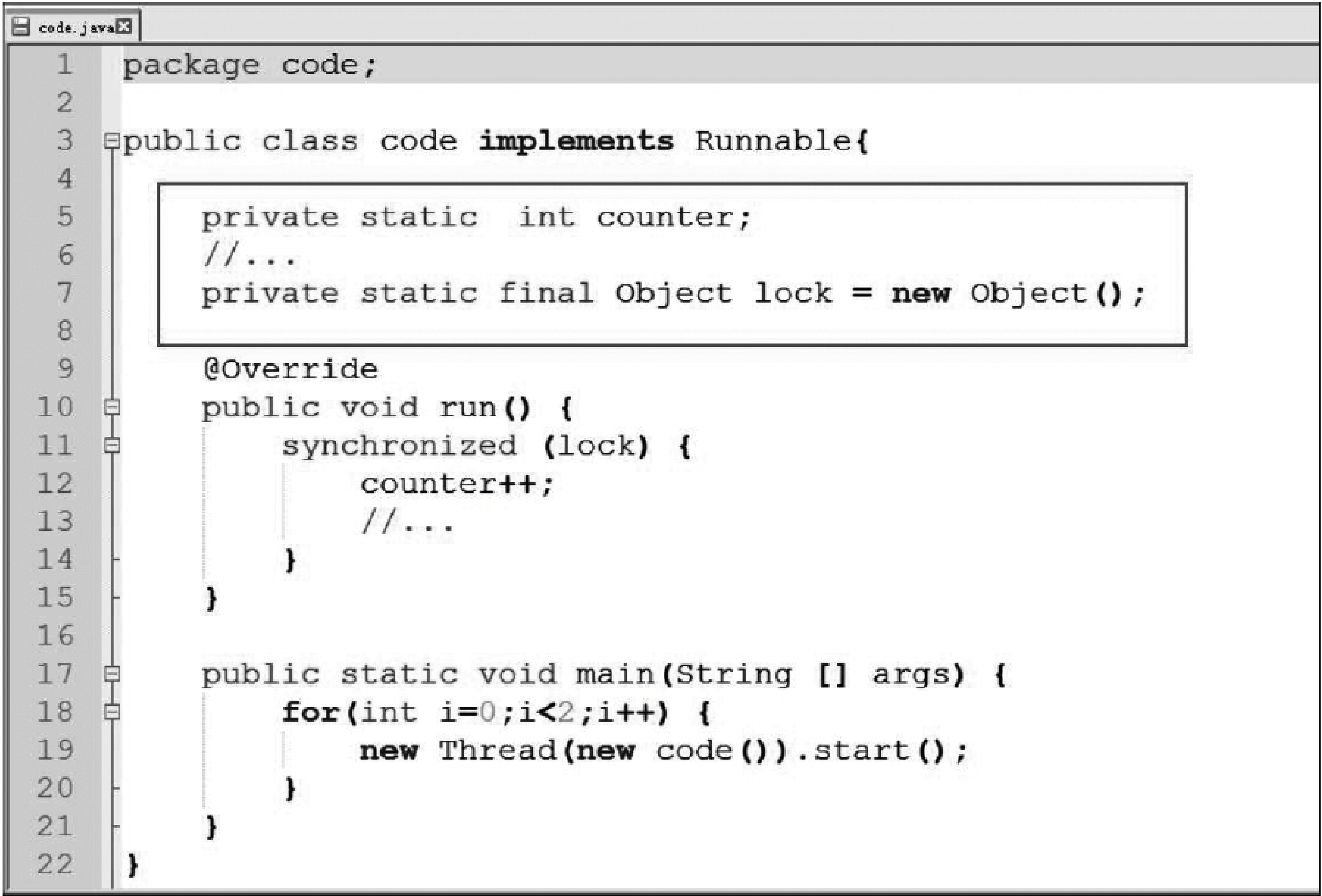
加固后的代码再次被检测后,对应的缺陷消失。



图 3-66　展开界面(3.2.6)

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-67 所示。



```
package code;

public class code implements Runnable{

    private static  int counter;
    //...
    private static final Object lock = new Object();

    @Override
    public void run() {
        synchronized (lock) {
            counter++;
            //...
        }
    }

    public static void main(String [] args) {
        for(int i=0;i<2;i++) {
            new Thread(new code()).start();
        }
    }
}
```

图 3-67　代码修改(3.2.6)

(2) 右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“不要使用实例锁来保护共享静态数据”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code-fix.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“缺陷总数”减少了,单击最右侧的“缺陷审计”按钮。

(7) 可以看到缺陷已经修复,如图 3-68 所示。



图 3-68 缺陷修复(3.2.6)

【实验思考】

(1) 什么是静态数据?

(2) 实例锁的作用是什么?

3.2.7 代码在循环中调用 wait()和 await()方法合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“不在循环中调用 wait()和 await()方法”的缺陷,并对发现的“不在循环中调用 wait()和 await()方法”的缺陷进行针对性的修复,确保所编写的代码中不存在“不在循环中调用 wait()和 await()方法”的缺陷。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个线程同步代码模块,使用 wait 进行线程等待,

需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：始终在循环中调用 wait()和 await()方法。

【实验原理】

Object.wait()方法会暂时地放弃所拥有的锁，这让其他线程有机会可以得到锁并继续执行。为了保证安全性，程序必须在 wait()方法返回之后对 while 循环的条件进行检测。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-69 所示。

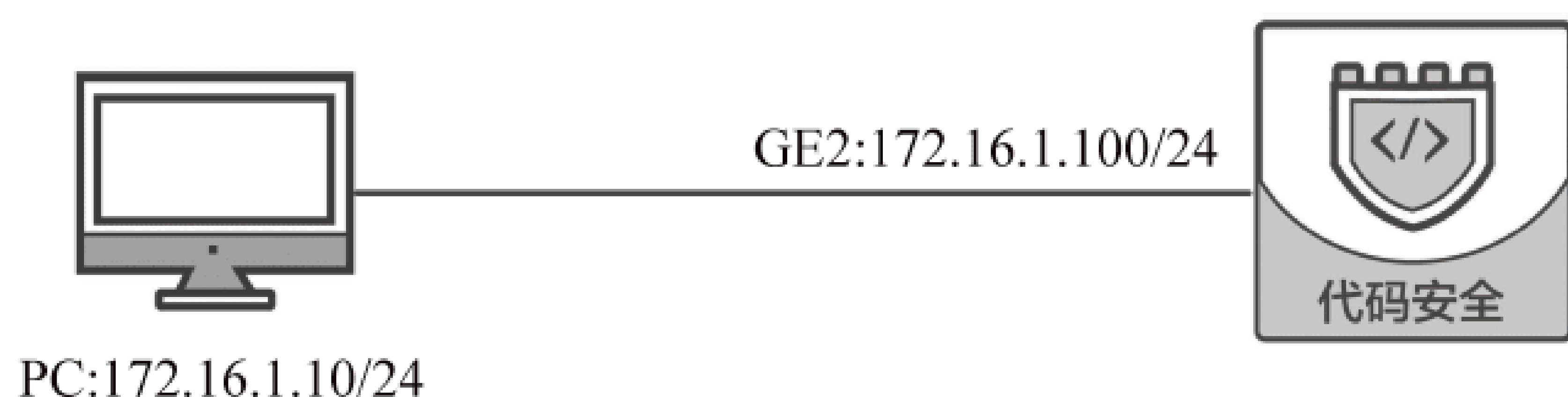


图 3-69　代码在循环中调用 wait()和 await()方法合规检测实验拓扑图

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-70 所示。

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“始终在循环中调用 wait 和 await 方法”，

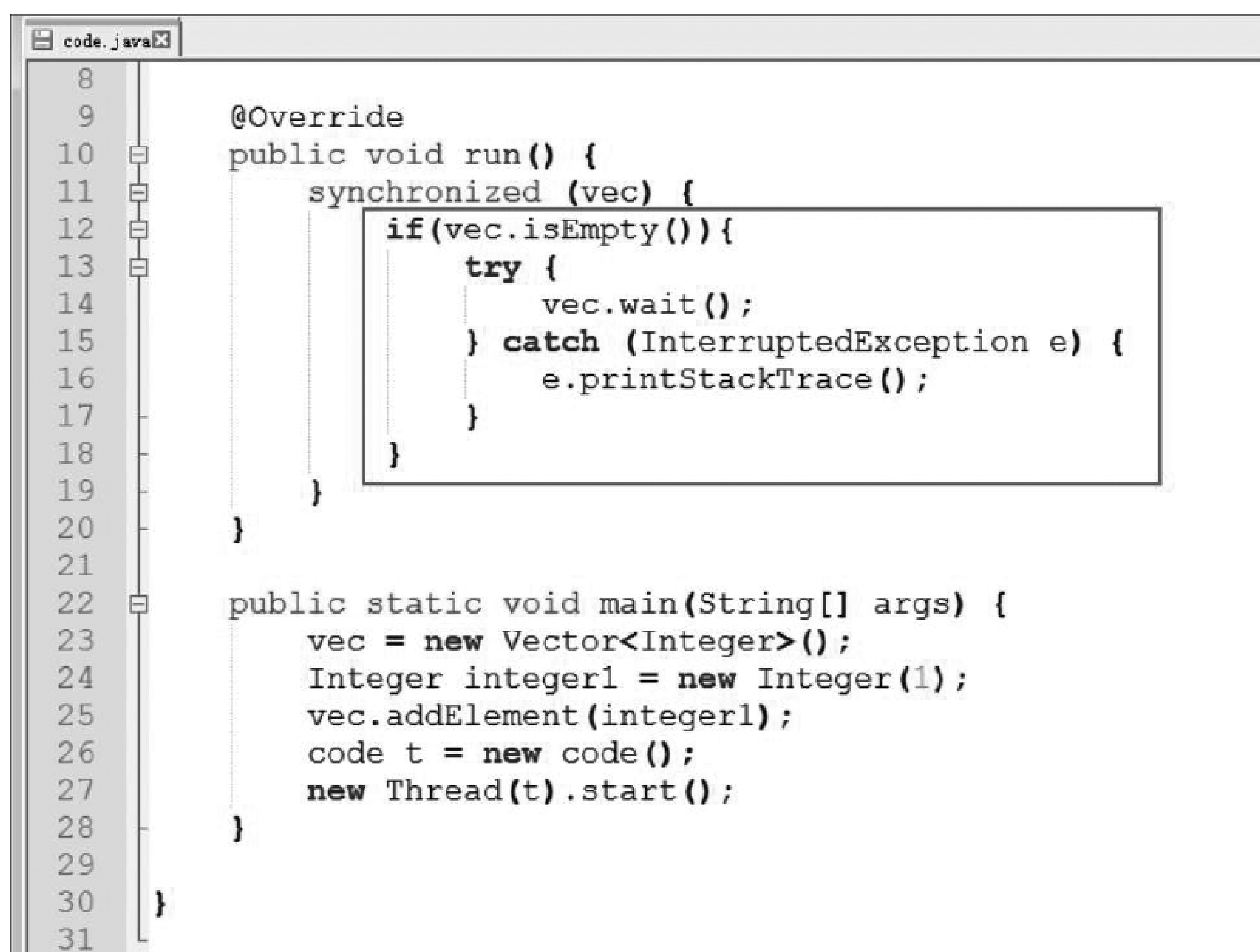


图 3-70　部分代码展示(3.2.7)

“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(9) 发起检测后,系统返回任务列表界面。当检测状态为“检测完成”时,单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷,并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录,如图 3-71 所示。



图 3-71　展开界面(3.2.7)

（11）选择 code.java 命令，快速定位有问题的代码。

（12）选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。

（13）选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后，对应的缺陷消失。

【实验结果】

（1）最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-72 所示。

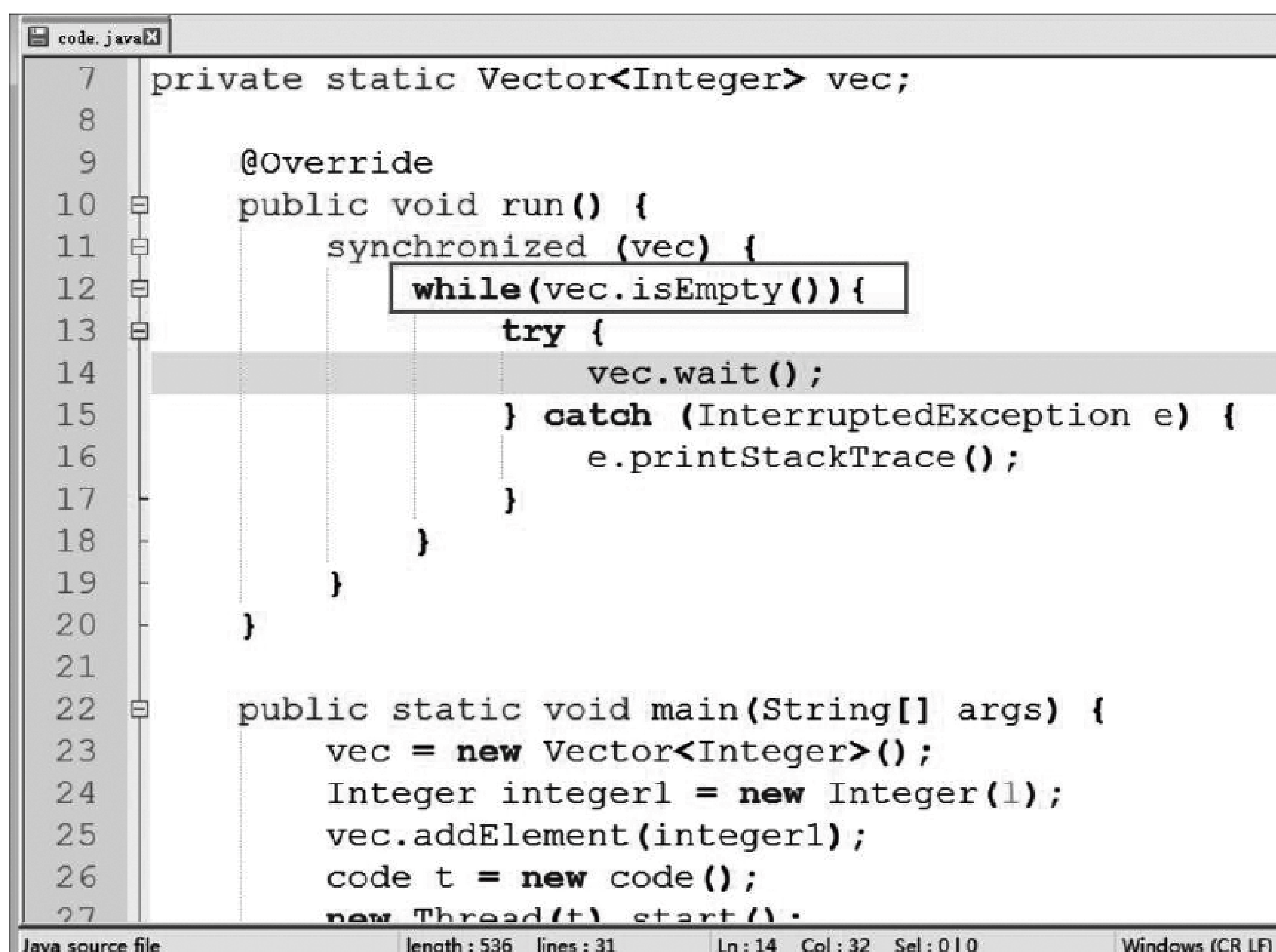


图 3-72　代码修改(3.2.7)

（2）右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

（3）“压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

（4）打开浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

（5）在配置信息界面中，“任务名称”输入“不要使用 Java 标准库已经公开的标识符修复检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

（6）在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮。

（7）可以看到缺陷已经修复，如图 3-73 所示。

【实验思考】

（1）在 Java 中唤醒线程的方法是什么？



图 3-73　缺陷修复(3.2.7)

(2) 什么是线程同步?

3.2.8　代码从流中读取的字符(或字节)和－1的区别合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“区分从流中读取的字符(或字节)和－1的不同”的缺陷,并对发现的“区分从流中读取的字符(或字节)和－1的不同”的缺陷进行针对性的修复,确保所编写的代码中不存在“区分从流中读取的字符(或字节)和－1的不同”的缺陷。

【知识点】

Java代码合规检测。

【场景描述】

A公司研发部工程师小王编写了一个读写文件代码模块,使用－1判断是否读取到了文件末尾,需要使用代码安全保障系统对编写的代码进行合规检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测:区分从流中读取的字符(或字节)和－1的不同。

【实验原理】

抽象方法InputStream.read()从一个输入源中读取一个简单字节,返回一个范围为0～255的整数。仅当在读取到输入流末尾的时候,它会返回－1。类似地Reader.read()方法读取一个单独的字符,返回范围为0～65535的整数。仅当在读取到输入流的末尾时,它会返回－1。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-74 所示。

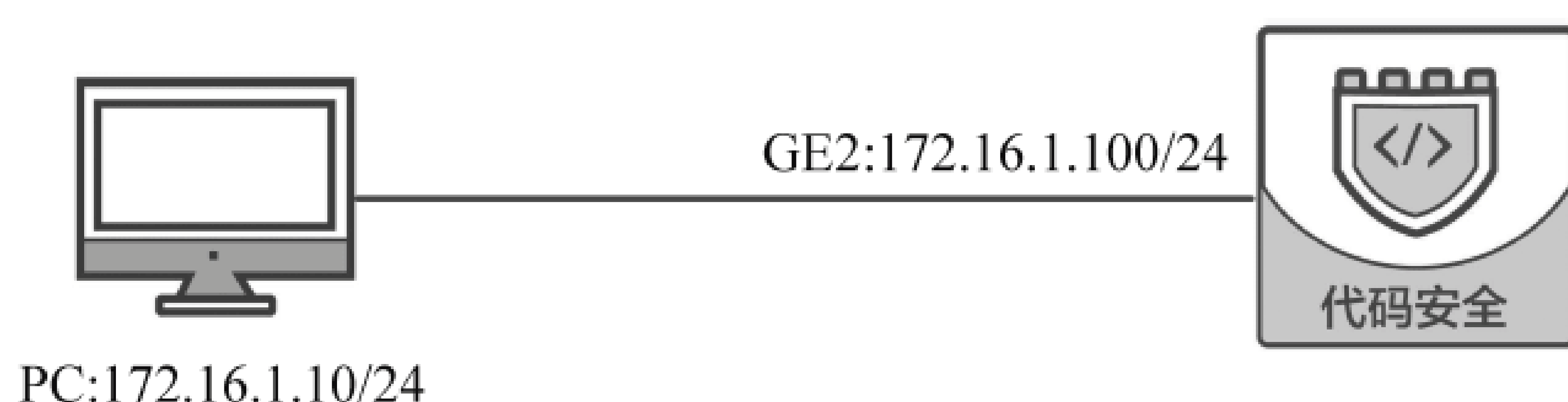


图 3-74　代码从流中读取的字符（或字节）和－1 的区别合规检测实验拓扑图

【实验思路】

（1）发起 Java 代码合规检测。

（2）根据修改建议对代码进行修改。

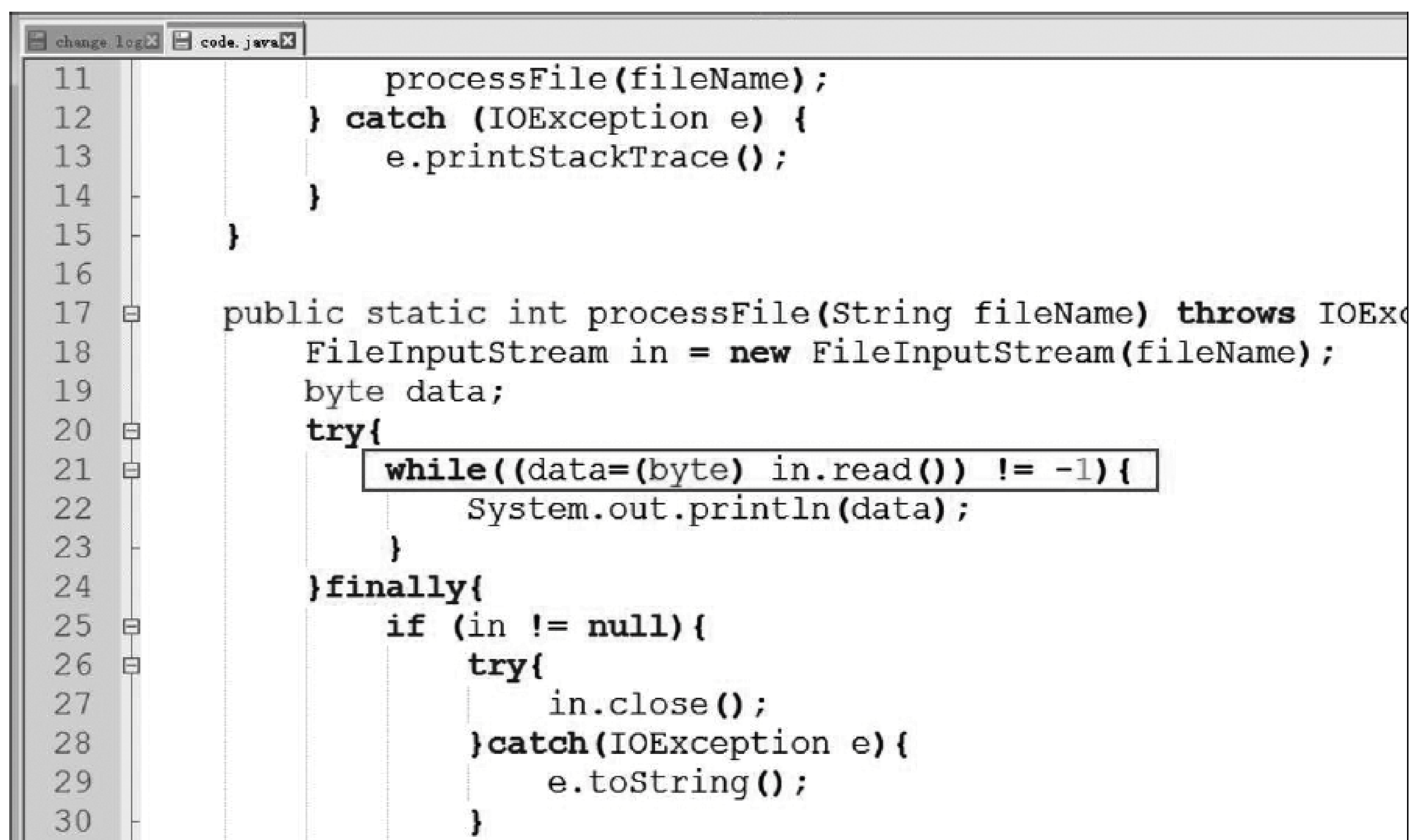
（3）检测修改效果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开“C:\code\code\src\code”目录下的 code.java 文件。

（3）程序中部分代码展示如图 3-75 所示。



```
11                 processFile(fileName);
12             } catch (IOException e) {
13                 e.printStackTrace();
14             }
15         }
16 
17         public static int processFile(String fileName) throws IOEx
18             FileInputStream in = new FileInputStream(fileName);
19             byte data;
20             try{
21                 while((data=(byte) in.read()) != -1){
22                     System.out.println(data);
23                 }
24             }finally{
25                 if (in != null){
26                     try{
27                         in.close();
28                     }catch(IOException e){
29                         e.toString();
30                     }
```

图 3-75　部分代码展示(3.2.8)

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“区分从流中读取的字符和-1 的不同”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录，如图 3-76 所示。



图 3-76 展开界面(3.2.8)

(11) 选择 code.java 命令，快速定位有问题的代码。

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后，对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-77 所示。

```
code.java
11              processFile(fileName);
12          } catch (IOException e) {
13              e.printStackTrace();
14          }
15      }
16
17      public static int processFile(String fileName) throws IOEx
18          FileInputStream in = new FileInputStream(fileName);
19          byte data;
20          int  inbuff;
21          try{
22              while((inbuff = in.read()) != -1){
23
24                  data   =   (byte)inbuff;
25                  System.out.println(data);
26              }
27          }finally{
28              if (in != null){
29                  try{
30                      in.close();
```

图 3-77　代码修改(3.2.8)

(2) 右击 code 文件夹,在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”,单击“立即压缩”按钮。

(4) 打开浏览器,在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“区分从流中读取的字符和−1 的不同”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code-fix.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“缺陷总数”减少了,单击最右侧的“缺陷审计”按钮。

(7) 可以看到缺陷已经修复,如图 3-78 所示。

【实验思考】

(1) 字符的定义是什么?

(2) 如果开发者过早地将读取的输入流返回的 int 类型转化为 byte 或 char 类型,还能用−1 来判断是否到达了输入流的结尾吗?

3.2.9　代码空无限循环合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“使用空的无限循环”的缺陷,并对发现的“使用空的无限循环”的缺陷进行针对性的修复,确保所编写的代码中不存在“使用空的无限循环”的缺陷。



图 3-78　缺陷修复(3.2.8)

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个文件导出代码模块，使用循环等待文件的生成，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要使用空的无限循环。

【实验原理】

一个空的无限循环会消耗大量的 CPU 计算资源而不做任何事情。优化编译器和即时系统(JIT)允许去除这样的循环。因此，程序不能包括这种没有代码的无限循环。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-79 所示。

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

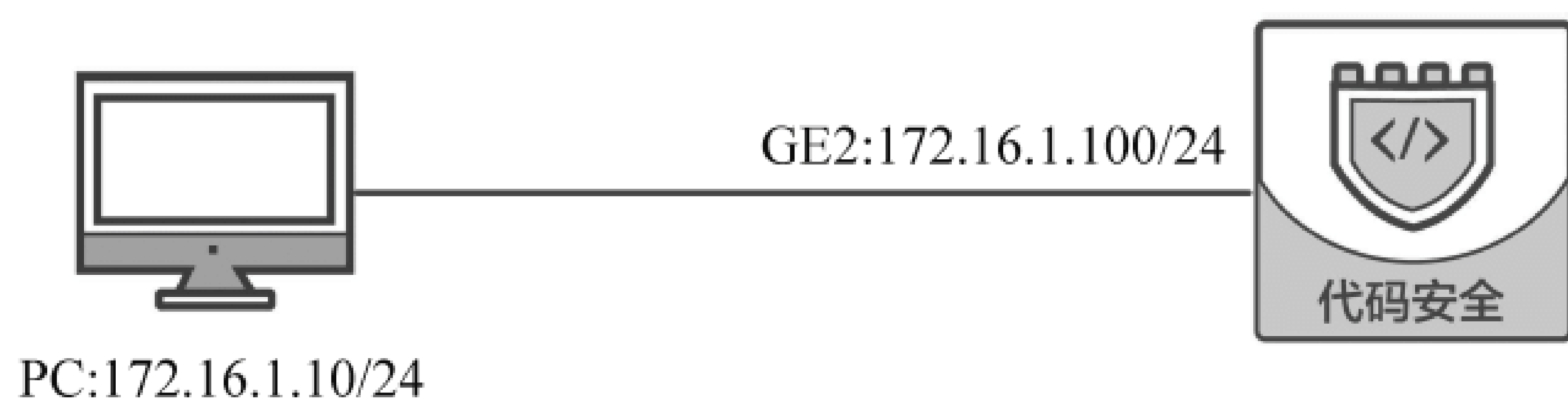


图 3-79　代码空无限循环合规检测实验拓扑图

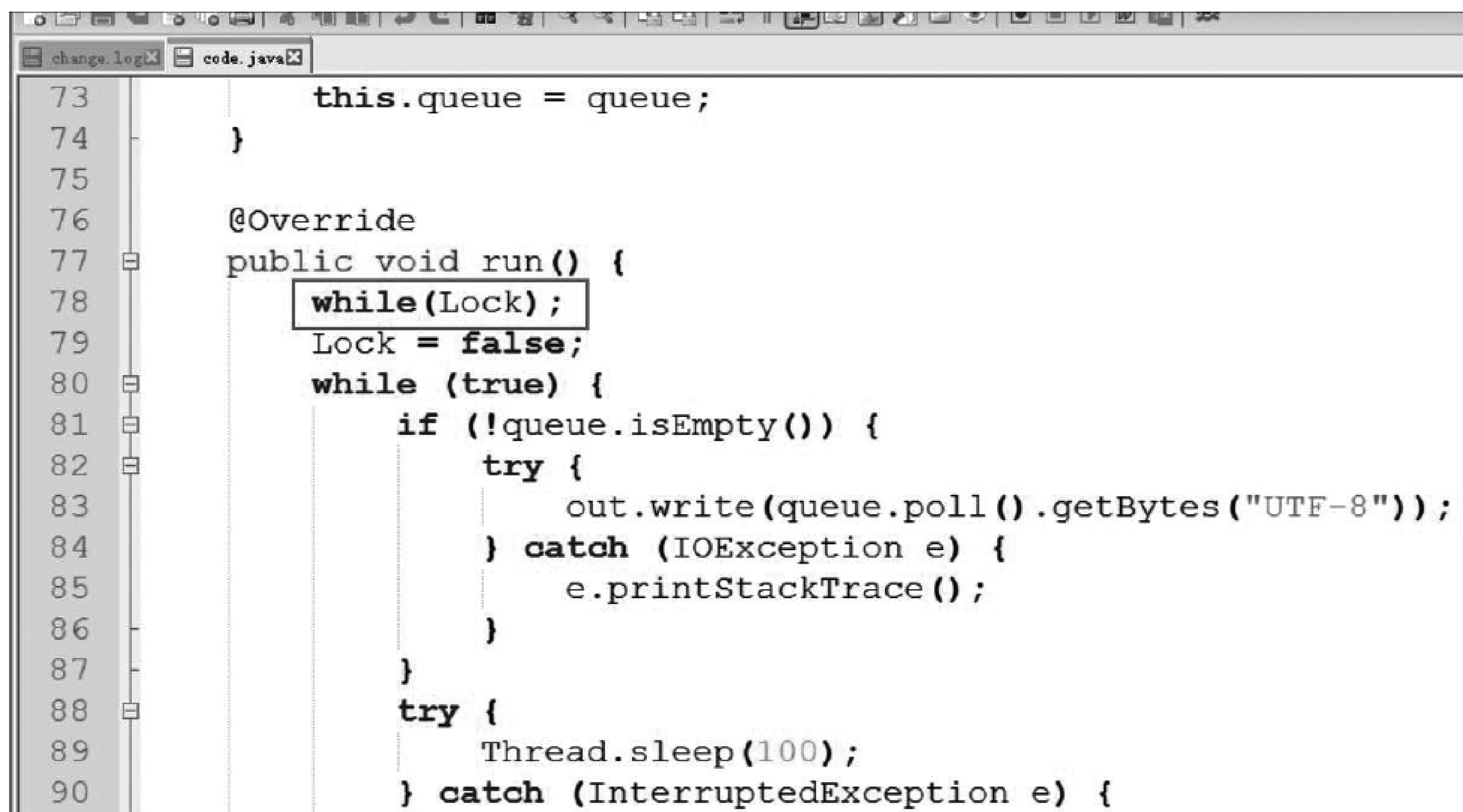
(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-80 所示。



```
73              this.queue = queue;
74          }
75
76          @Override
77          public void run() {
78              while(Lock);
79              Lock = false;
80              while (true) {
81                  if (!queue.isEmpty()) {
82                      try {
83                          out.write(queue.poll().getBytes("UTF-8"));
84                      } catch (IOException e) {
85                          e.printStackTrace();
86                      }
87                  }
88                  try {
89                      Thread.sleep(100);
90                  } catch (InterruptedException e) {
```

图 3-80　部分代码展示(3.2.9)

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要使用空的无限循环”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单周“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧

的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷,并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录,如图 3-81 所示。



图 3-81 展开界面(3.2.9)

(11) 选择 code.java 命令,快速定位有问题的代码。

(12) 选择“详细信息”命令,可以查看问题代码中该条缺陷的具体信息。

(13) 选择“修复建议”命令,查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测后,对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口,打开“C:\code\code\src\code”目录下的 code.java 文件,根据修复建议对代码进行修改,保存修改并关闭文件,如图 3-82 所示。

(2) 右击 code 文件夹,在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”,单击“立即压缩”按钮。

(4) 打开浏览器,在之前的界面上选择“快速检测”→“+发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“不要使用空的无限循环”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的“code-fix.zip”文件,取消勾选“缺陷检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“缺陷总数”减少了,单击最右侧的“缺陷审计”按钮。

(7) 可以看到缺陷已经修复,如图 3-83 所示。

```
76          @Override
77          public void run() {
78              synchronized(queue){
79                  while (true) {
80                      if (!queue.isEmpty()) {
81                          try {
82                              out.write(queue.poll().getBytes("UTF-8"));
83                          } catch (IOException e) {
84                              e.printStackTrace();
85                          }
86                      }
87                      try {
88                          Thread.sleep(100);
89                      } catch (InterruptedException e) {
90                          e.printStackTrace();
91                      }
92                  }
93              }
94          }
95
96  }
97
```

图 3-82　代码修改(3.2.9)



图 3-83　缺陷修复(3.2.9)

【实验思考】

(1) 怎样理解 while 循环?

(2) while 循环和 do-while 循环的区别是什么?

3.2.10 代码析构函数合规检测实验

【实验目的】

通过使用代码安全保障系统检测文件上传代码模块是否存在“使用析构函数”的违规情况，并对发现的“使用析构函数”的违规情况进行针对性的修复，确保所编写的代码中不存在“使用析构函数”的违规情况。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个资源管理代码模块，使用析构函数进行了资源释放操作，需要使用代码安全保障系统对编写的代码进行合规检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的合规检测：不要使用析构函数。

【实验原理】

析构函数的执行提供了一种方式来释放资源，例如流、文件和网络连接，但是析构函数还存在很多问题，比如析构函数的执行时没有固定时间的，这是由 JVM 所决定的。因此，在析构函数中调用时效性的功能(例如关闭文件句柄)是有问题的。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开发人员检测出源代码中的缺陷并进行针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 3-84 所示。

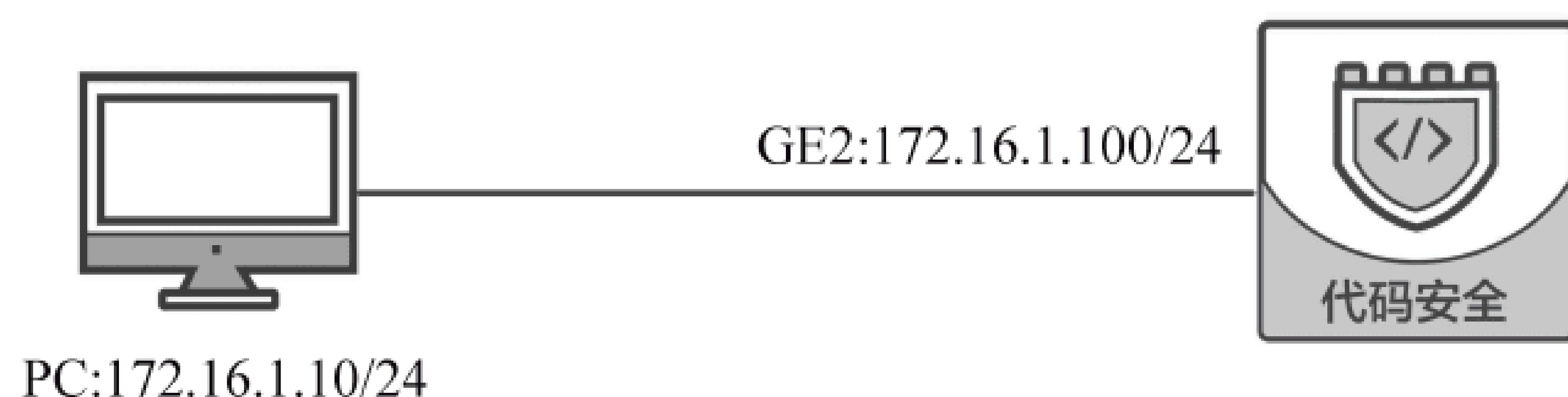


图 3-84　代码析构函数合规检测实验拓扑图

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 根据修改建议对代码进行修改。

(3) 检测修改效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 打开“C:\code\code\src\code”目录下的 code.java 文件。

(3) 程序中部分代码展示如图 3-85 所示。

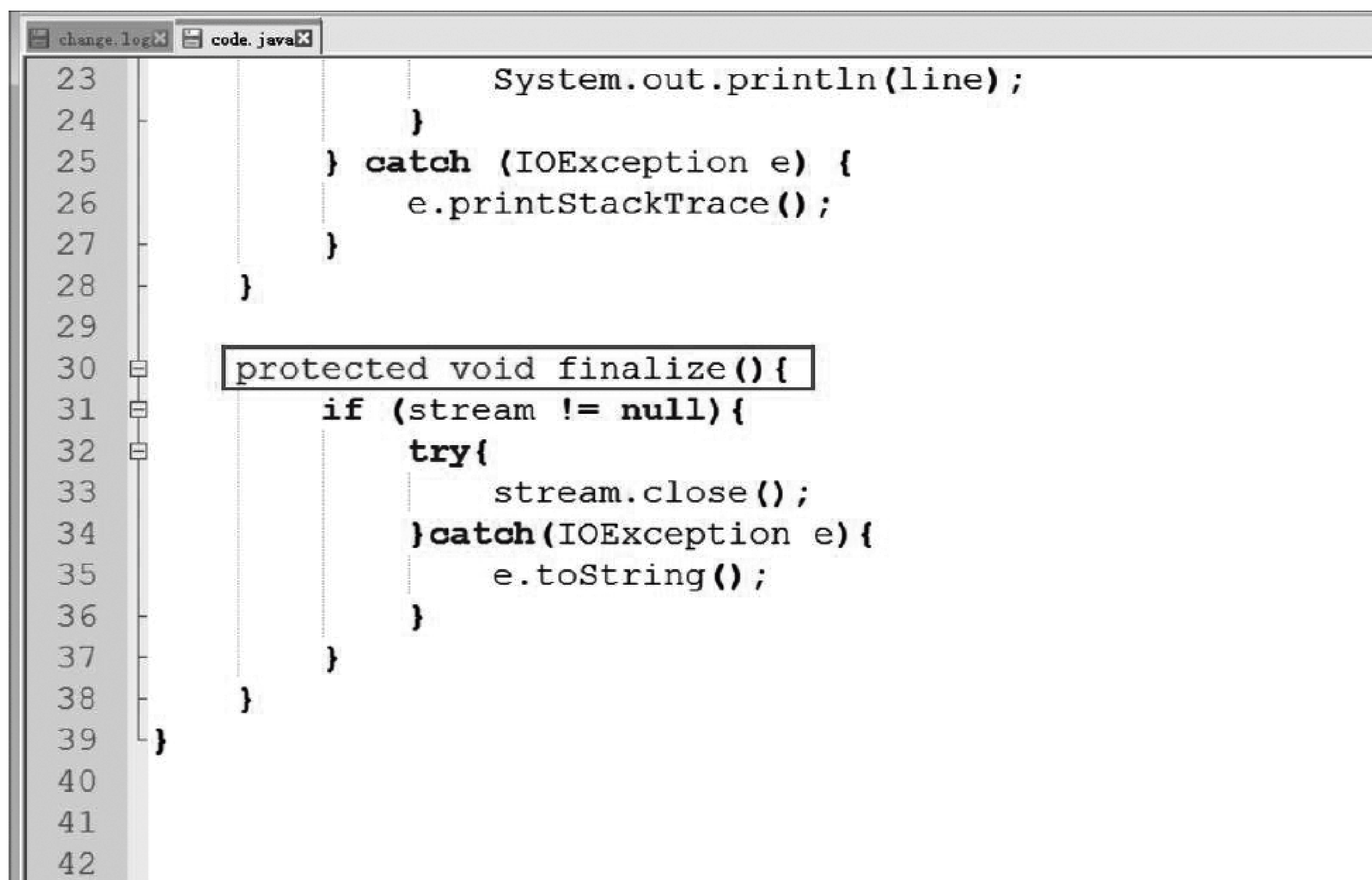


图 3-85　部分代码展示(3.2.10)

(4) 打开 Chrome 浏览器，输入代码卫士登录网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(5) 单击“继续前往 172.16.1.100”按钮。

(6) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(7) 进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

(8) 在配置信息界面中，“任务名称”输入“不要使用析构函数”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(9) 发起检测后，系统返回任务列表界面。当检测状态为“检测完成”时，单击最右侧的“缺陷审计”按钮。

(10) 可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“中”。单击左侧框中的“+”按钮展开目录，如图 3-86 所示。

(11) 选择“code.java(31)”命令，快速定位有问题的代码。

(12) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。



图 3-86　展开界面(3.2.10)

(13) 选择“修复建议”命令，查看代码卫士给的修复建议。

【实验预期】

加固后的代码再次被检测，对应的缺陷消失。

【实验结果】

(1) 最小化浏览器窗口，打开“C:\code\code\src\code”目录下的 code.java 文件，根据修复建议对代码进行修改，保存修改并关闭文件，如图 3-87 所示。

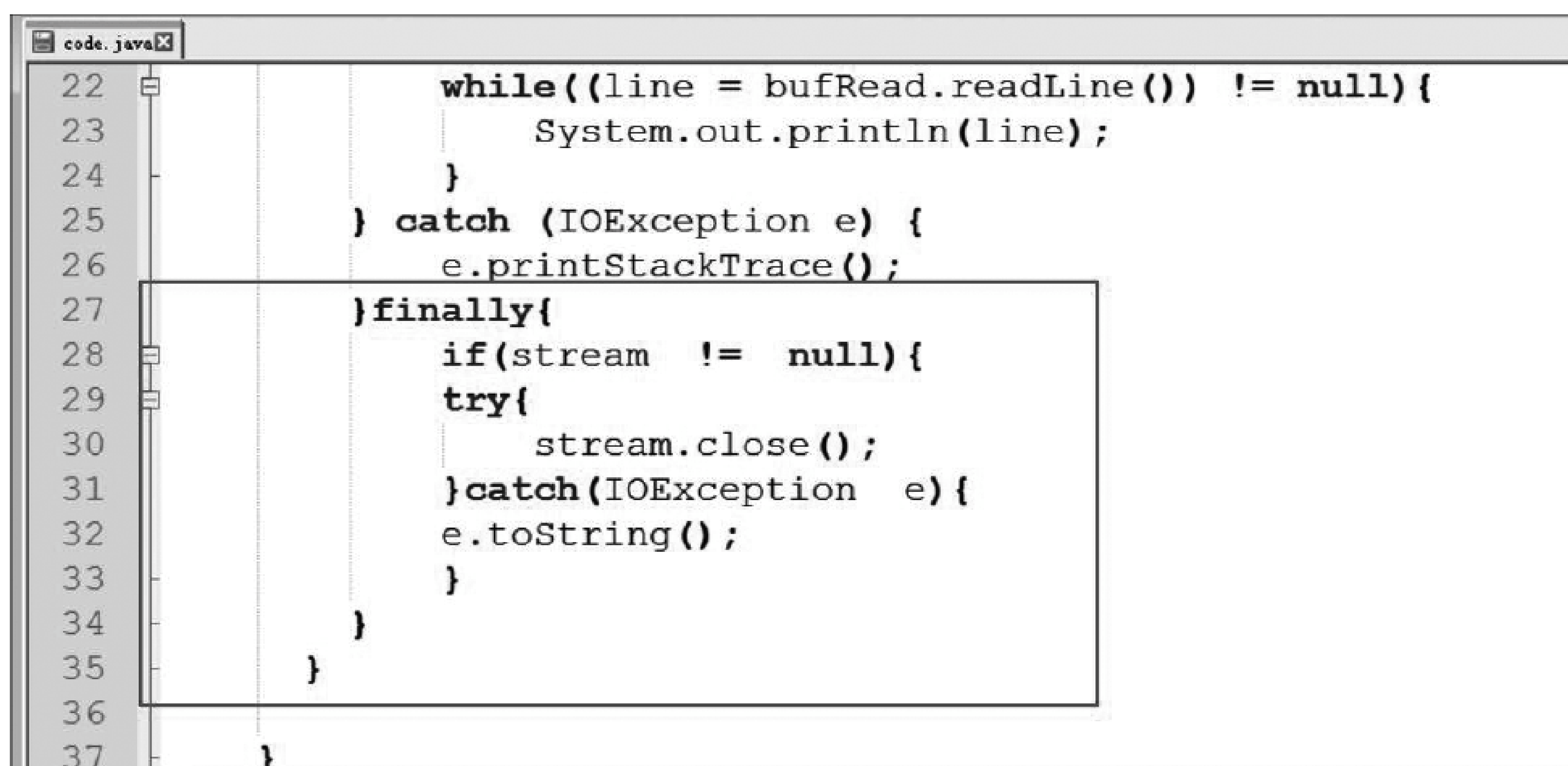


图 3-87　代码修改(3.2.10)

(2) 右击 code 文件夹，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开浏览器，在之前的界面上选择“快速检测”→“+发起快速检测”命令。

（5）在配置信息界面中，“任务名称”输入“不要使用析构函数”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code-fix.zip”文件，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

（6）在任务列表界面等待检测完成，可以看到该项目检测完成后，“缺陷总数”减少了，单击最右侧的“缺陷审计”按钮。

（7）可以看到缺陷已经修复，如图 3-88 所示。



图 3-88　缺陷修复(3.2.10)

【实验思考】

（1）什么是析构函数？

（2）何时调用析构函数？

第4章 代码安全保障系统发起检测任务

代码安全保障系统的基本理念是使组织以用最小的代价将源代码安全检测融入已有开发和测试流程中，并提供灵活的定制化接口，满足组织的个性化定制需求。代码安全保障系统支持本地源代码检测，也可支持从 SVN 等代码管理系统中获取源代码进行检测。

本章主要介绍通过代码安全保障系统发起缺陷检测、合规检测以及溯源检测等实验，帮助管理员或者用户更好地熟悉代码安全保障系统的基本使用方法，通过检测源代码中存在哪些安全漏洞，最大限度地降低源代码安全风险。

4.1 发起 C/C++检测任务

4.1.1 C 语言缺陷检测任务实验

【实验目的】

在代码安全保障系统上发起本地代码的缺陷检测任务。

【知识点】

代码卫士、缺陷检测。

【场景描述】

A 公司研发部工程师小王编写了一个代码模块，此模块会将项目文件导出为.zip 文件并存储在本地，现在需要使用代码安全保障系统对导出的.zip 文件进行安全类缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。请帮助小王完成本地代码上传的缺陷检测任务。

【实验原理】

奇安信代码安全保障系统设置了一系列详细的代码检测规则，检测系统要求只能上传.zip 文件，通过奇安信前端程序可以将本地的项目文件转换为合规格的.zip 文件，将此文件上传到代码卫士平台，通过设置相关参数就可以发起代码缺陷检测任务。

【实验设备】

- 安全设备：代码安全保障系统 1 套。

• 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-1 所示。

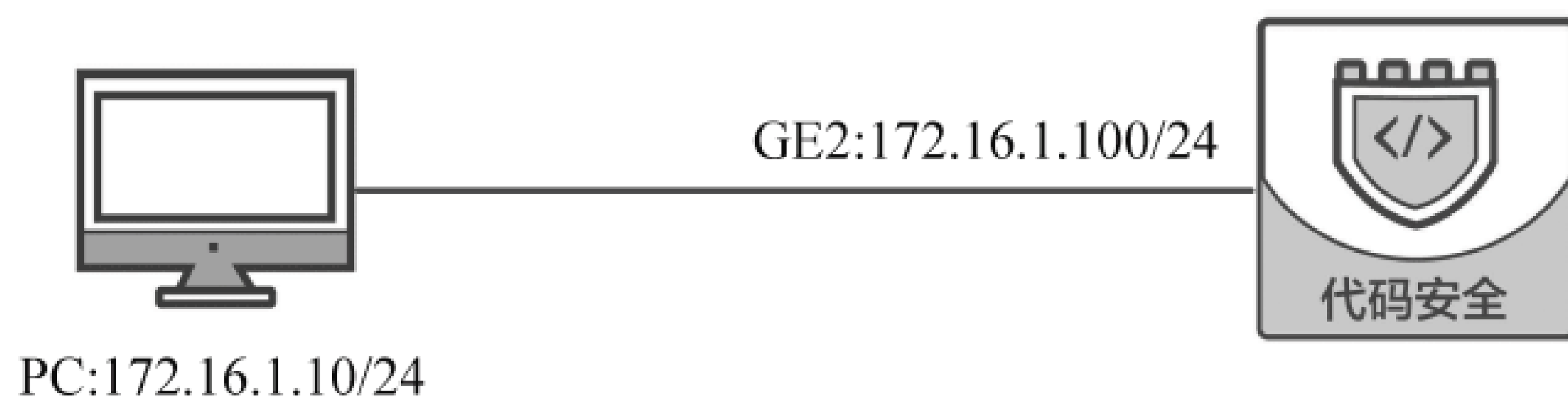


图 4-1　C 语言缺陷检测任务实验拓扑图

【实验思路】

(1) 建立工程。

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，发起 C 语言缺陷检测任务。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击打开 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”选项框右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.cpp，单击“添加”按钮。

(7) 打开“C:\”下的 code.txt，复制其中的代码，如图 4-2 所示。

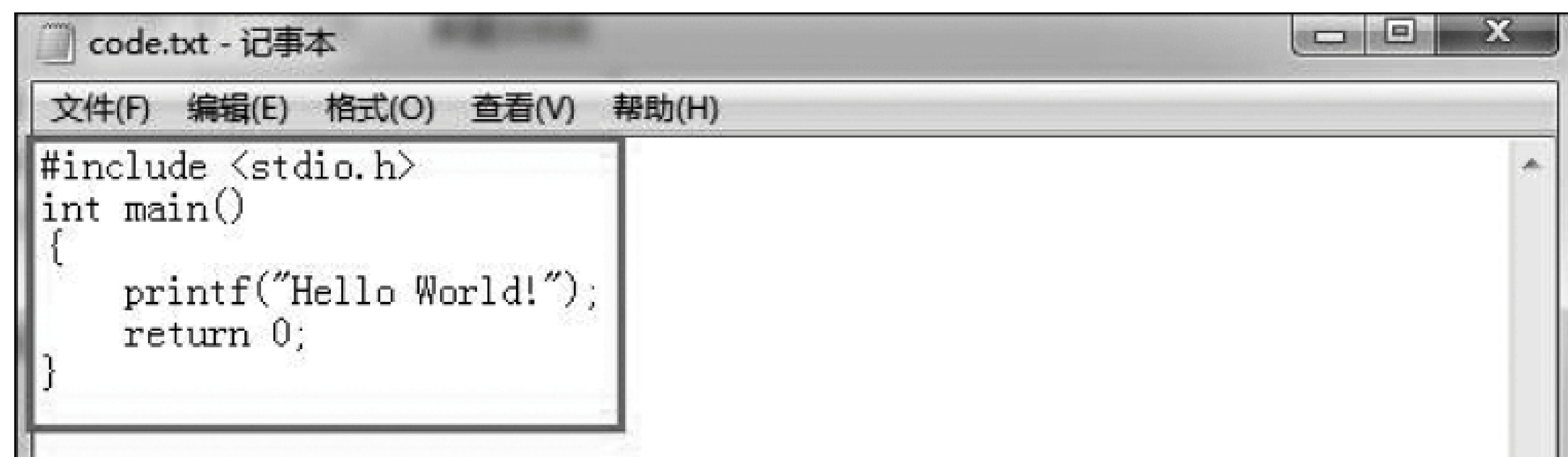


图 4-2　复制代码(4.1.1)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

(9) 弹出项目编译确认对话框，单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(11) 进入“管理员：VS 2015 开发人员命令提示”界面，在命令行中输入命令“manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\ ”下的 codemanager 文件夹，显示中间文件 testmanager.zip 生成成功。

(13) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(14) 在“高级”界面中单击“添加例外”按钮。

(15) 界面跳转到代码卫士登录界面，“用户名”输入 admin，“密码”输入“admin123!@#”，单击“登录”按钮。

(16) 选择“快速检测”→“＋发起快速检测”命令。

(17) “任务名称”输入“发起 C 语言缺陷检测任务实验”，“开发语言”选中“C/C++”单选钮，勾选“缺陷检测模板[C/C++]”复选框。

(18) 选择“浏览”命令。

(19) 选择“本地磁盘(C:)”命令，选择右侧的 codemanager，单击“打开”按钮。

(20) 选择文件 codemanager，单击“打开”按钮。

(21) 返回“快速检测”界面，单击“发起检测”按钮。

(22) 界面跳转到“缺陷检测”列表，成功发起缺陷检测。

【实验预期】

(1) 在缺陷检测列表查看任务“发起 C 语言缺陷检测任务实验”。

(2) 导出缺陷检测报告。

【实验结果】

1. 在缺陷检测列表查看任务“发起 C 语言缺陷检测任务实验”

进入代码卫士“快速检测”界面，查看任务“发起 C 语言缺陷检测任务实验”的检测结果，如图 4-3 所示。

2. 导出缺陷检测报告

(1) 单击任务“发起 C 语言缺陷检测任务实验”右侧的“检测报告导出”按钮，如图 4-4 所示。

(2) 弹出导出报告设置窗口，默认设置，单击“导出报告”按钮，如图 4-5 所示。

(3) 跳转到“报告管理”界面，找到任务“发起 C 语言缺陷检测任务实验”的检测报告，单击“下载”按钮即可获得检测报告，如图 4-6 所示。

【实验思考】

(1) 自己编写程序，发起 C 语言缺陷检测任务。

(2) 导出检测报告，查看代码其他问题。



图 4-3　查看检测结果(4.1.1)



图 4-4　导出检测报告

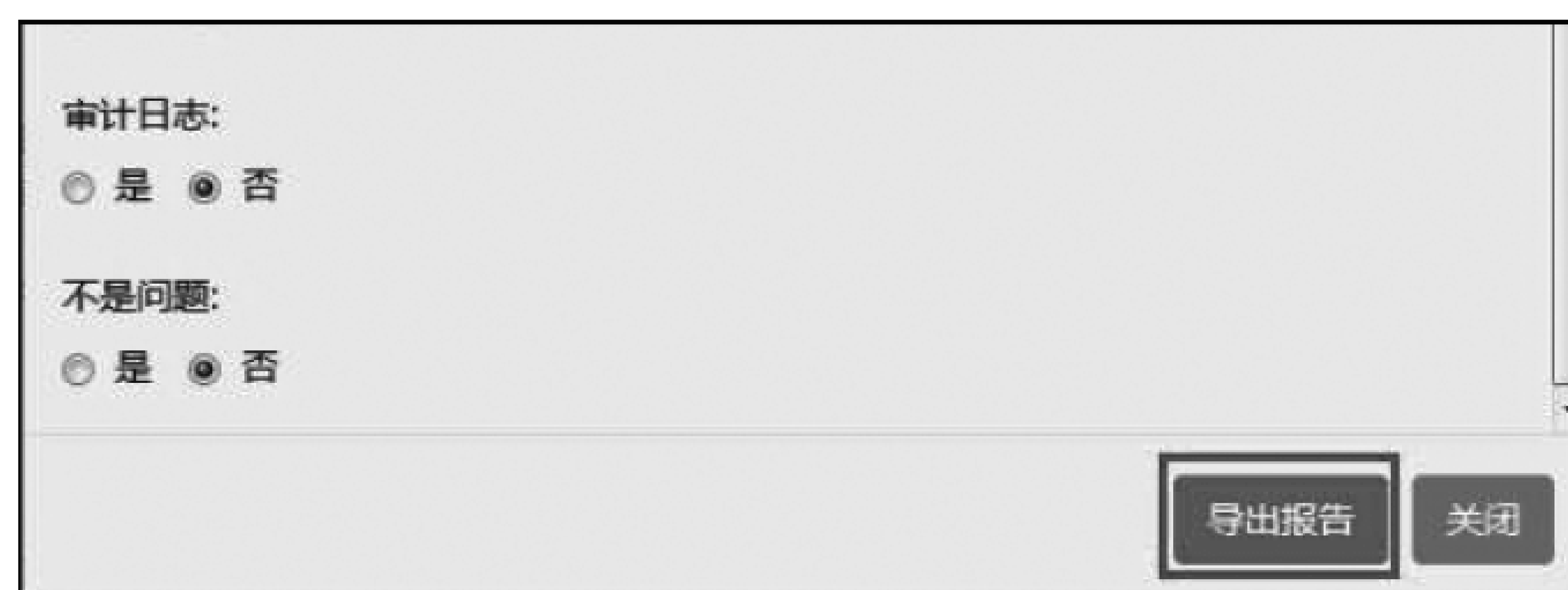


图 4-5　继续导出检测报告

图 4-6　获得检测报告

4.1.2　C 语言合规检测任务实验

【实验目的】

在代码安全保障系统上发起本地代码的合规检测任务。

【知识点】

代码卫士、合规检测。

【场景描述】

A 公司研发部工程师小王编写了一个代码模块，此模块会将项目文件导出为.zip 文件并存储在本地，现在需要使用代码安全保障系统对导出的.zip 文件进行违规类缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。请帮助小王完成本地代码上传的合规检测任务。

【实验原理】

奇安信代码安全保障系统设置了一系列详细的代码检测规则，检测系统要求只能上传.zip 文件，通过奇安信前端程序可以将本地的项目文件转换为合规格的.zip 文件，将此文件上传到代码卫士平台，通过设置相关参数就可以发起代码合规检测任务。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-7 所示。

【实验思路】

(1) 建立工程。

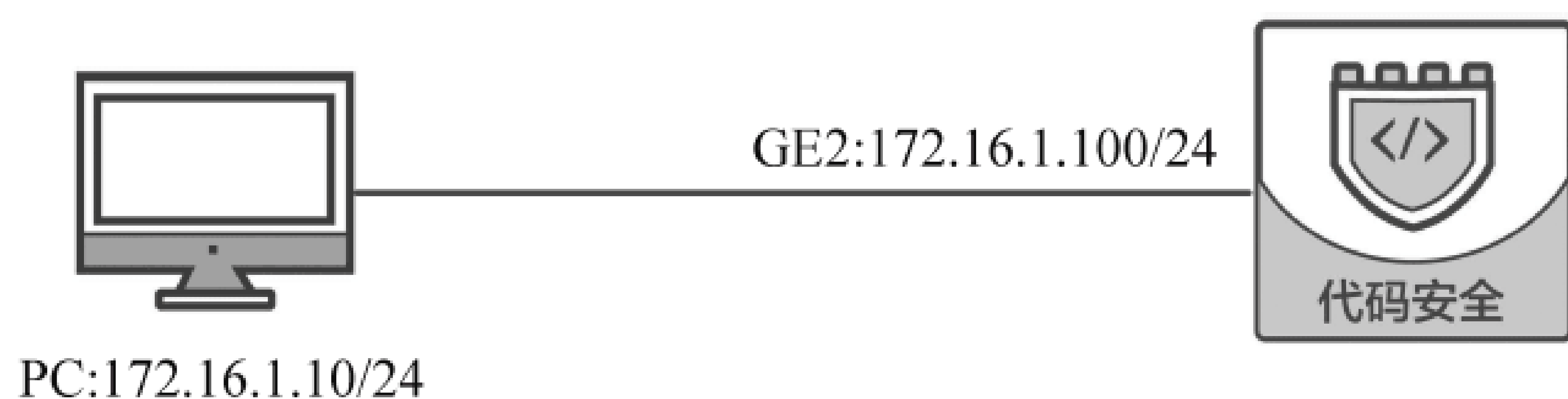


图 4-7　C 语言合规检测任务实验拓扑图

(2) 生成代码的中间表示文件。

(3) 上传到代码卫士，发起 C 语言合规检测任务。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”按钮，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”选项框右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 再次进入 Visual Studio 2015 主界面，查看界面上“解决方案资源管理器”小窗口，已经新建好 code 项目。右击“源文件”，在弹出的快捷菜单中选择“添加”→“新建项”命令，添加一个新项。

(6) 进入“添加新项”界面，左侧“已安装”默认选择“Visual C++”命令，在界面中间选择“C++文件(.cpp)”命令。在下侧“名称”选项框内输入 code.cpp，选择“添加”命令。

(7) 打开“C:\ ”下的 code.txt，复制其中的代码，如图 4-8 所示。

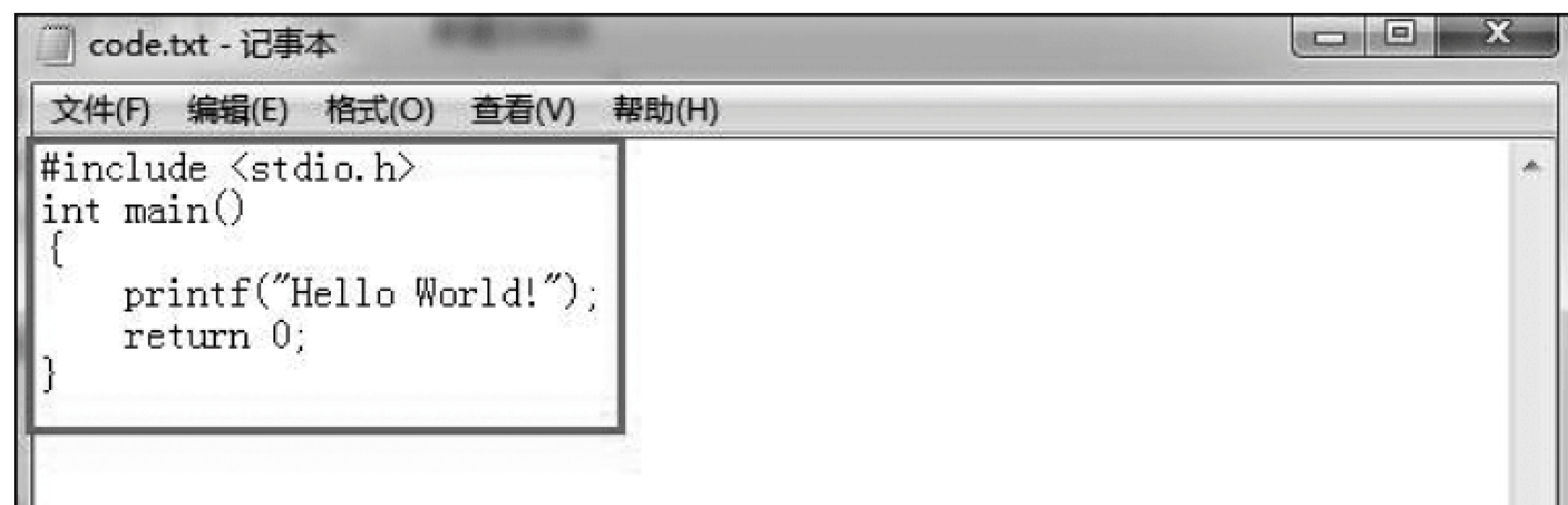


图 4-8　复制代码(4.1.2)

(8) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“本地 Windows 调试器”按钮。

(9) 弹出项目编译确认对话框，单击“是”按钮。

(10) 选择“开始”→“VS 2015 开发人员命令提示”命令，打开命令行界面。

(11) 进入“管理员: VS 2015 开发人员命令提示”界面，在命令行中输入命令“manager -o C:\codemanager devenv C:\code\code.sln /rebuild”。

(12) 进入终端机“C:\ ”下的 codemanager 文件夹，显示中间文件 testmanager.zip 生成成功。

(13) 打开 Chrome 浏览器,网址输入"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(14) 在"高级"界面中单击"添加例外"按钮。

(15) 跳转到代码卫士登录界面,"用户名"输入 admin,"密码"输入"admin123!@#",单击"登录"按钮。

(16) 选择"快速检测"→"+发起快速检测"命令。

(17) "任务名称"输入"发起 C 语言合规检测任务实验","开发语言"选中"C/C++"单选钮,勾选"缺陷检测模板[C/C++]"复选框。

(18) 选择"浏览"命令。

(19) 选择"本地磁盘(C:)"命令,选择右侧的 codemanager,单击"打开"按钮。

(20) 选择文件 codemanager,单击"打开"按钮。

(21) 返回"快速检测"界面,单击"发起检测"按钮。

(22) 界面跳转到"缺陷检测"列表,成功发起缺陷检测。

【实验预期】

(1) 在合规检测列表查看任务"发起 C 语言合规检测任务实验"。

(2) 导出合规检测报告。

【实验结果】

1. 在合规检测列表查看任务"发起 C 语言合规检测任务实验"

进入代码卫士"快速检测"界面,查看任务"发起 C 语言合规检测任务实验"的检测结果,如图 4-9 所示。



图 4-9 查看检测结果(4.1.2)

2. 导出合规检测报告

(1) 单击任务"发起 C 语言合规检测任务实验"右侧的"检测报告导出"按钮。

(2) 弹出导出报告设置窗口,默认设置,单击"导出报告"按钮。

（3）界面跳转到“报告管理”界面，找到任务“发起 C 语言合规检测任务实验”的检测报告，单击“下载”按钮即可获得检测报告。

【实验思考】

（1）自己编写程序，发起 C 语言合规检测任务。

（2）导出检测报告，查看代码其他问题。

4.1.3　C# 语言缺陷检测任务实验

【实验目的】

在代码安全保障系统上发起 C# 本地代码的缺陷检测任务。

【知识点】

代码卫士、缺陷检测、C#。

【场景描述】

A 公司研发部工程师小王编写的代码使用了 svn（代码托管）服务管理，现在要通过代码安全保障系统对编写的 C# 代码进行安全测试，以便了解代码安全保障系统的基本使用方法。请帮助小王完成 C# 代码模块的快速检测。

【实验原理】

奇安信代码安全保障系统设置了一系列详细的代码检测规则，检测系统要求只能上传.zip 文件，通过奇安信前端程序可以将本地的项目文件转换为合规格的.zip 文件，将此文件上传到代码卫士平台，通过设置相关参数就可以发起代码缺陷检测任务。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：安装有 VS 2015 和 Chrome 浏览器的 Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-10 所示。

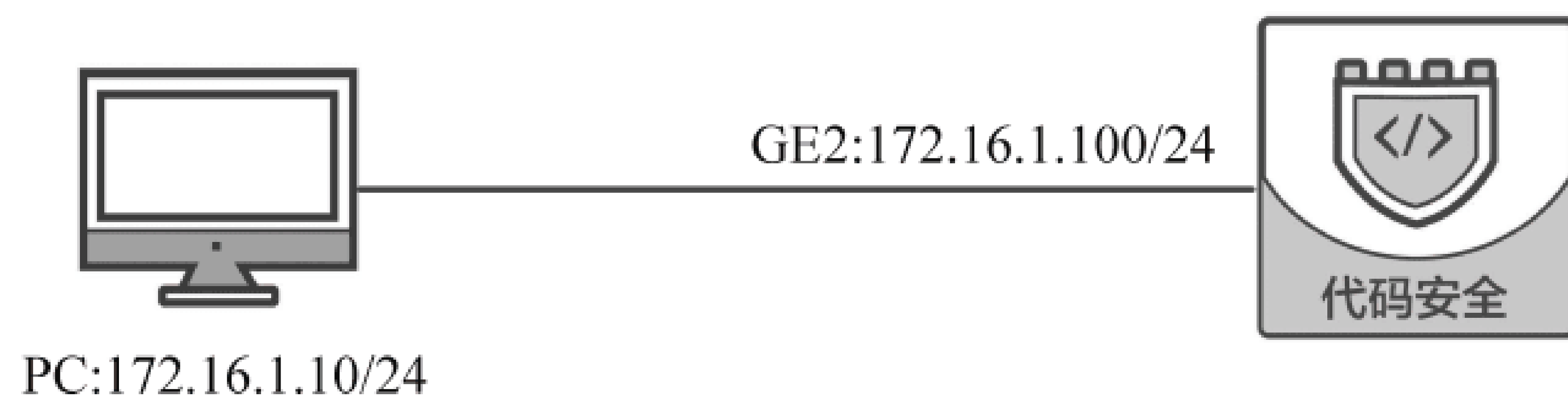


图 4-10　C# 语言缺陷检测任务实验拓扑图

【实验思路】

（1）建立工程。

（2）压缩项目文件。

（3）上传到代码卫士，发起 C# 语言缺陷检测任务。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 在终端机桌面双击 Visual Studio 2015，显示主界面。

(3) 进入主界面，选择“文件”→“新建”→“项目”命令，新建一个项目。

(4) 进入“新建项目”界面，在左侧“模板”选择“Visual C++”命令，界面中间选项选择“空项目”，下侧“名称”选项框内输入 code，单击“位置”选项右侧的“浏览”按钮，选择“C:\”，其他默认选择。单击“确定”按钮。

(5) 打开“C:\ ”下的 code.txt，复制其中的代码。

(6) 进入 Visual Studio 2015 代码编辑界面，在编辑框内粘贴代码，单击“启动”按钮，运行结束后关闭 VS 2015，如图 4-11 所示。

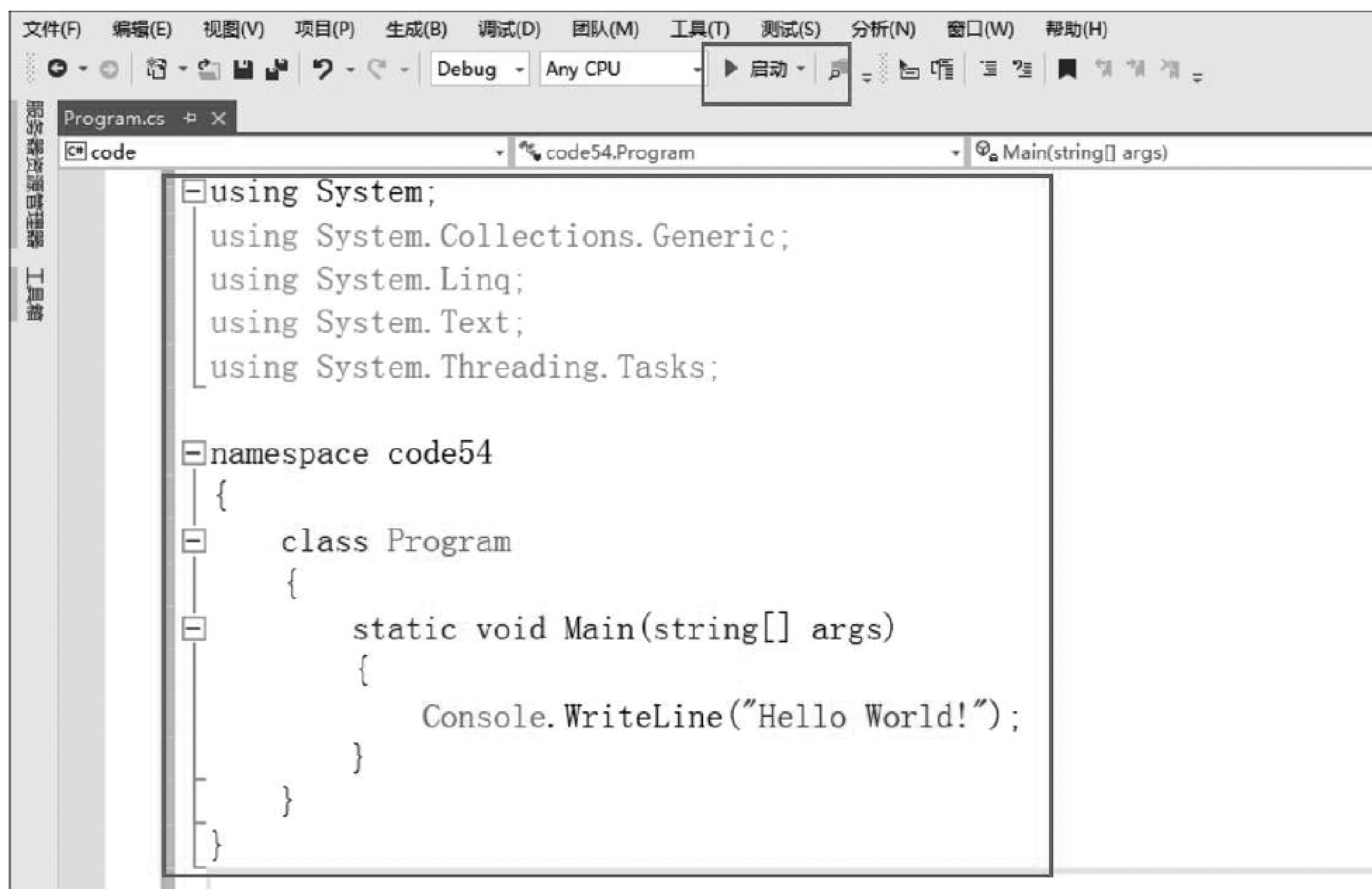


图 4-11　代码编辑界面(4.1.3)

(7) 选择“开始”→“360 安全中心”命令，右击“360 压缩”，在弹出的快捷菜单中选择“以管理员身份运行”命令，如图 4-12 所示。

(8) 弹出运行程序确认对话框，单击“是(Y)”按钮，如图 4-13 所示。

(9) 在压缩窗口选择“计算机”命令，如图 4-14 所示。

(10) 选择“本地磁盘(C:)”命令，如图 4-15 所示。

(11) 选择 code→“添加”命令，如图 4-16 所示。

(12) 弹出压缩确认窗口，单击“立即压缩”按钮，如图 4-17 所示。

(13) 打开 Chrome 浏览器，网址输入“https://172.16.1.100”，在“您的连接不是私密连接”中单击“高级”按钮。



图 4-12　打开“压缩”界面



图 4-13　以管理员权限运行压缩

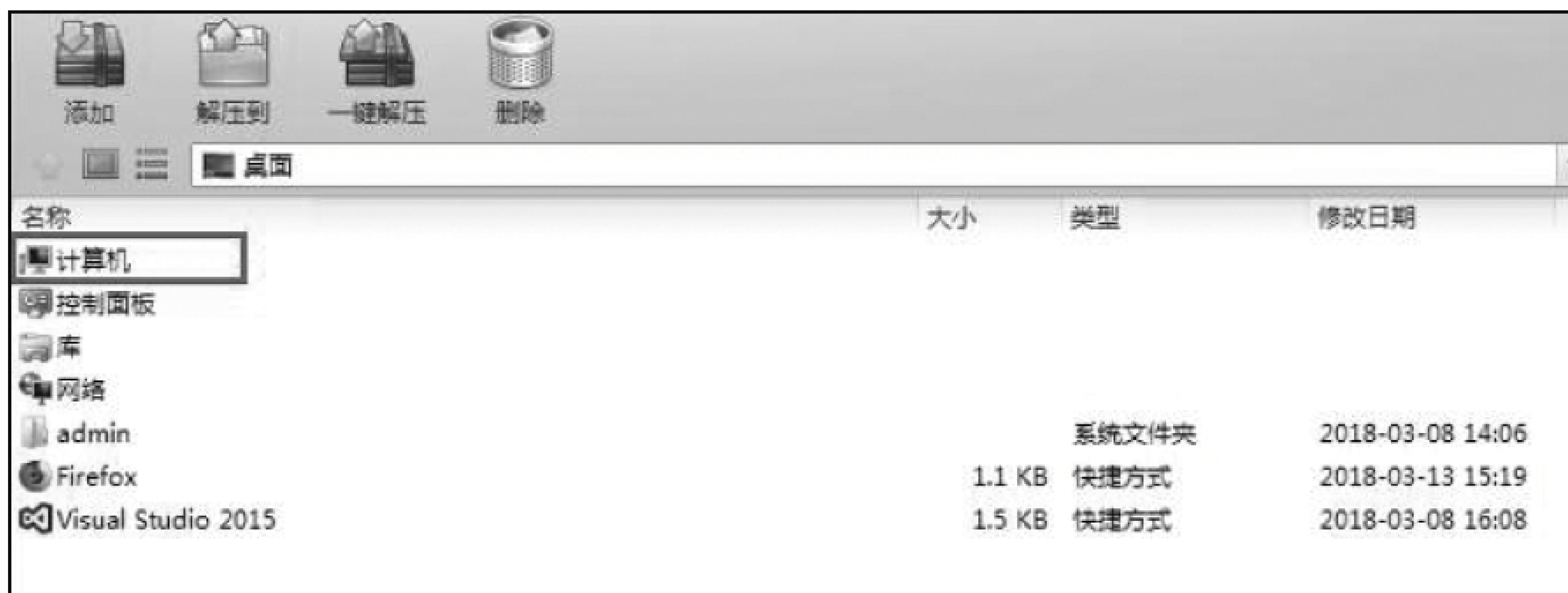


图 4-14　选择“计算机”命令



图 4-15　选择“本地磁盘(C:)”命令

(14) 在“高级”界面中单击“继续前往 172.16.1.100(不安全)”按钮。

(15) 跳转到代码卫士登录界面,“用户名”输入 admin,“密码”输入“admin123!@#”,单击“登录”按钮。

(16) 选择“快速检测”→“+发起快速检测”命令。

(17) “任务名称”输入“发起 Csharp 语言缺陷检测任务实验”,“开发语言”选中“C#”单选钮。

(18) 选择“浏览”命令。



图 4-16　选择文件



图 4-17　压缩文件

(19) 选择“本地磁盘(C:)”命令，单击右侧的 code，单击“打开”按钮。

(20) 返回“快速检测”界面，单击“发起检测”按钮。

(21) 界面跳转到“缺陷检测”列表，成功发起缺陷检测。

【实验预期】

(1) 在缺陷检测列表查看任务“发起 Csharp 语言缺陷检测任务实验”。

(2) 导出缺陷检测报告。

【实验结果】

1. 在缺陷检测列表查看任务“发起 Csharp 语言缺陷检测任务实验”

进入代码卫士“快速检测”界面，查看任务“发起 Csharp 语言缺陷检测任务实验”的检测结果，如图 4-18 所示。

2. 导出缺陷检测报告

(1) 单击任务“发起 Csharp 语言缺陷检测任务实验”右侧的“检测报告导出”按钮。

(2) 弹出导出报告设置窗口，默认设置，单击“导出报告”按钮。

(3) 界面跳转到“报告管理”界面，找到任务“发起 Csharp 语言缺陷检测任务实验”的检测报告，单击“下载”按钮，即可获得检测报告。

【实验思考】

(1) 自己编写程序发起 C# 语言缺陷检测任务。

(2) 导出检测报告，查看代码其他问题。



图 4-18　查看检测结果(4.1.3)

4.2　发起 PHP 检测任务

4.2.1　PHP 语言缺陷检测任务实验

【实验目的】

通过使用代码安全保障系统检测 Git 服务器中所存放的代码是否含有编码缺陷，了解代码安全保障系统的基本使用方法。

【知识点】

发起 PHP 语言缺陷检测任务。

【场景描述】

A 公司研发部工程师小王编写的代码使用 Git 服务管理，需要使用代码安全保障系统对编写的 PHP 代码进行安全测试，了解代码安全保障系统的基本使用方法。请帮助小王完成 PHP 代码模块的快速检测。

【实验原理】

Git 是一个分布式版本控制系统，由 Linux 开源社区开发。此版本控制系统可以使项目的设计者将设计恢复到之前任一状态的选择权，这种选择权在设计过程无法进行下去时特别重要。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。
- Git 服务器：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-19 所示。

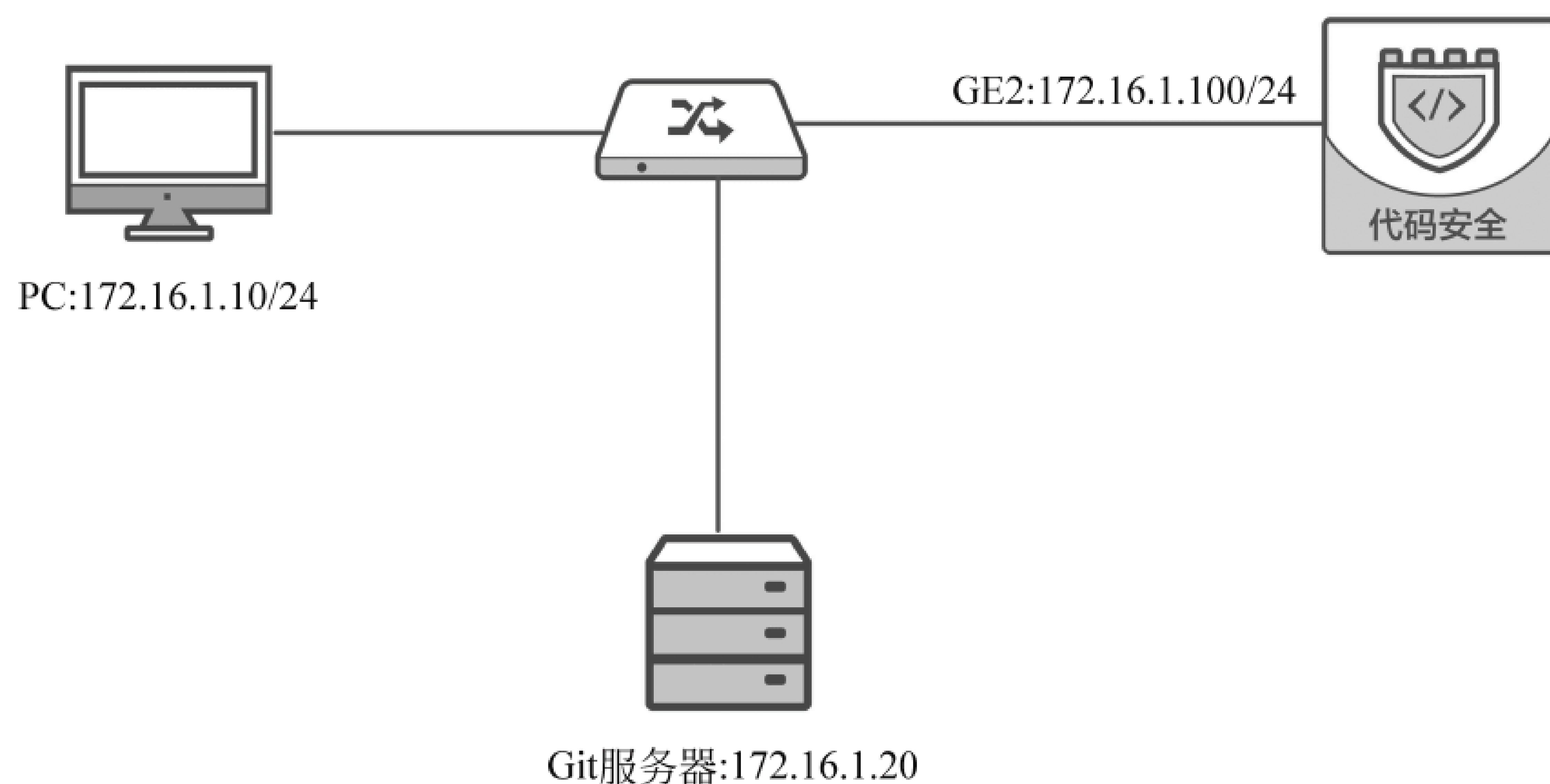


图 4-19　PHP 语言缺陷检测任务实验拓扑图

【实验思路】

发起 PHP 代码缺陷检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧主机终端 PC 虚拟机,如需登录密码,输入 123456,如图 4-20 所示。

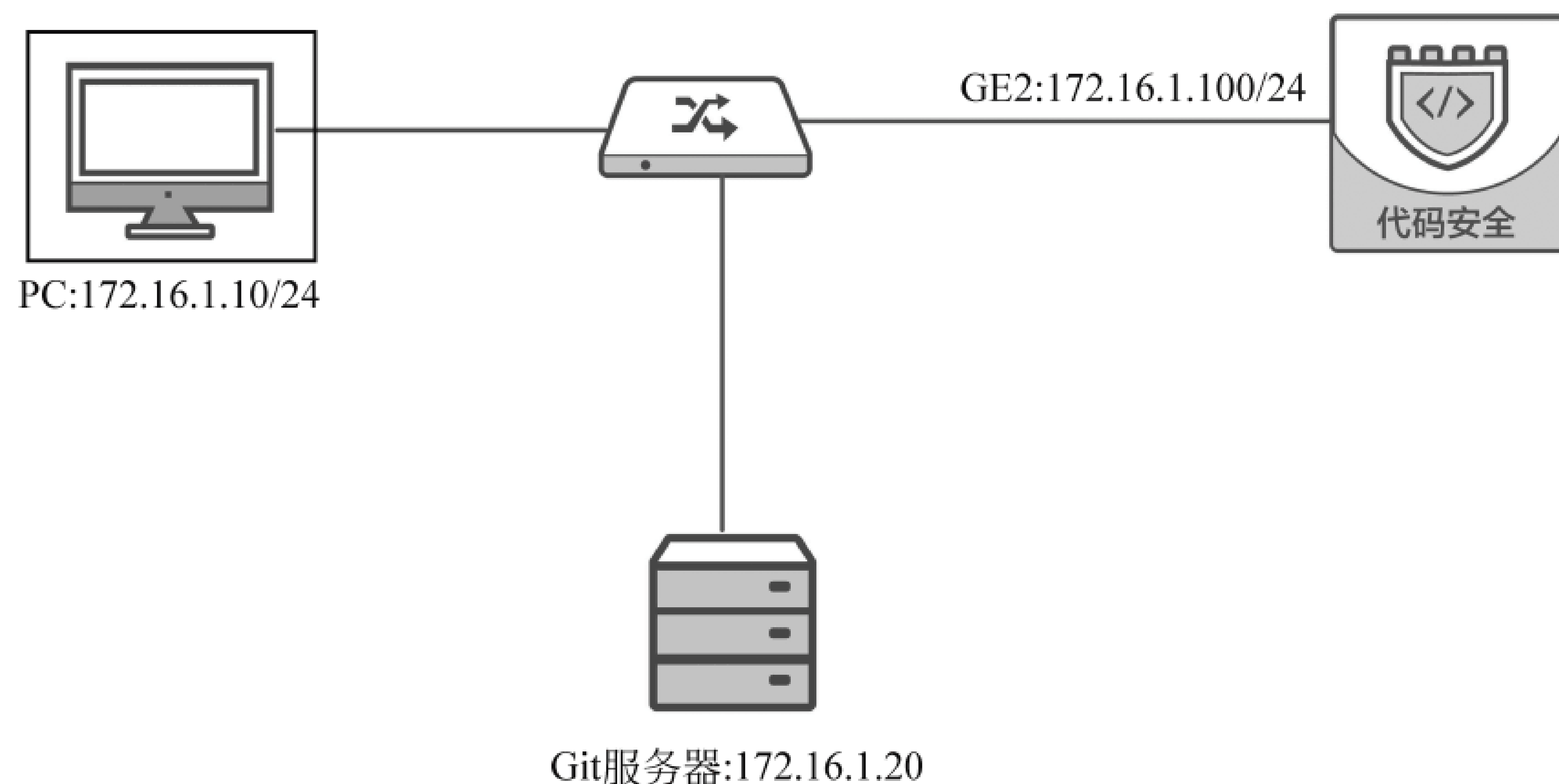


图 4-20　实验平台对应拓扑图

(2) 打开 Chrome 浏览器,输入代码卫士的 IP 地址"https://172.16.1.100",在显示的"您的连接不是私密连接"界面中单击"高级"按钮。

(3) 单击"继续前往 172.16.1.100(不安全)"按钮。

(4) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(5) 进入代码卫士后,选择"快速检测"→"+发起快速检测"命令。

(6) 配置信息界面"任务名称"输入"检测 Git 服务器代码","开发语言"选中 PHP 单选钮,"源码来源"设置为 Git,"Git 地址"输入"http://172.16.1.20:1000/r/test.git","Git 分支名称"默认为 master,"Git 用户名"为 admin,"Git 密码"为 admin,其他选择默认配置,单击"发起检测"按钮。

【实验预期】

代码安全保障系统可以检测出 Git 服务器中的代码缺陷。

【实验结果】

发起检测后代码安全保障系统会从 Git 服务器上面取出后缀为.php 类型的文件进行检测,检测完成后可以查看缺陷的数量以及等级(红色代表高危),如图 4-21 所示。

图 4-21　检测结果(4.2.1)

【实验思考】

(1) 如何降低代码的缺陷等级?

(2) 如何将代码上传到 Git 服务器?

4.2.2　代码安全保障系统发起持续任务检测实验

【实验目的】

通过使用代码卫士对 Git 服务器中的代码按照周期性进行持续性缺陷检测,使学生掌握对代码按周期进行持续性缺陷检测的方法。

【知识点】

发起持续检测任务。

【场景描述】

A 公司研发部编写的代码使用 Git 服务管理,需要使用代码安全保障系统对提交的代码按照周期进行持续性缺陷检测。请帮助完成代码的持续性检测任务。

【实验原理】

项目管理可以定义检测任务的执行频率和执行周期,支持定期执行检测任务,每执行

一次，称为一个检测批次。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。
- Git 服务器：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-22 所示。

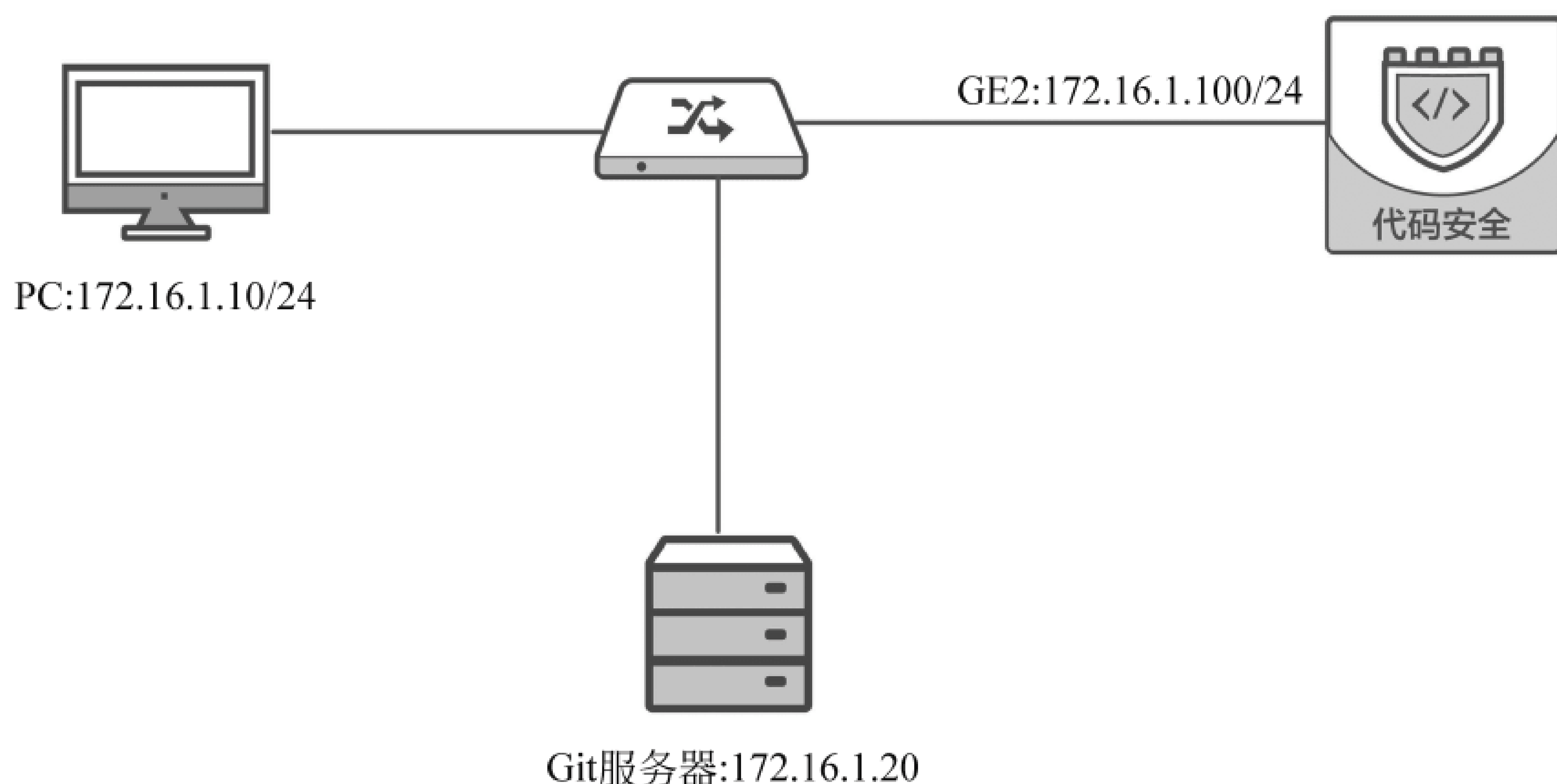


图 4-22　代码安全保障系统发起持续任务检测实验拓扑图

【实验思路】

(1) 将代码上传至远程 Git 仓库。

(2) 发起持续检测任务。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧 PC 主机终端虚拟机，如需登录密码，输入 123456。

(2) 进入"C:\code"目录，该目录下有实验需要用到的 code.php 代码文件。

(3) 在"C:\code"目录下右击，在弹出的快捷菜单中选择"Git Bash Here"命令，进入 Git 命令行界面。

(4) 输入命令"git add code.php"，将 code.php 文件的信息添加到索引库中，如图 4-23 所示。

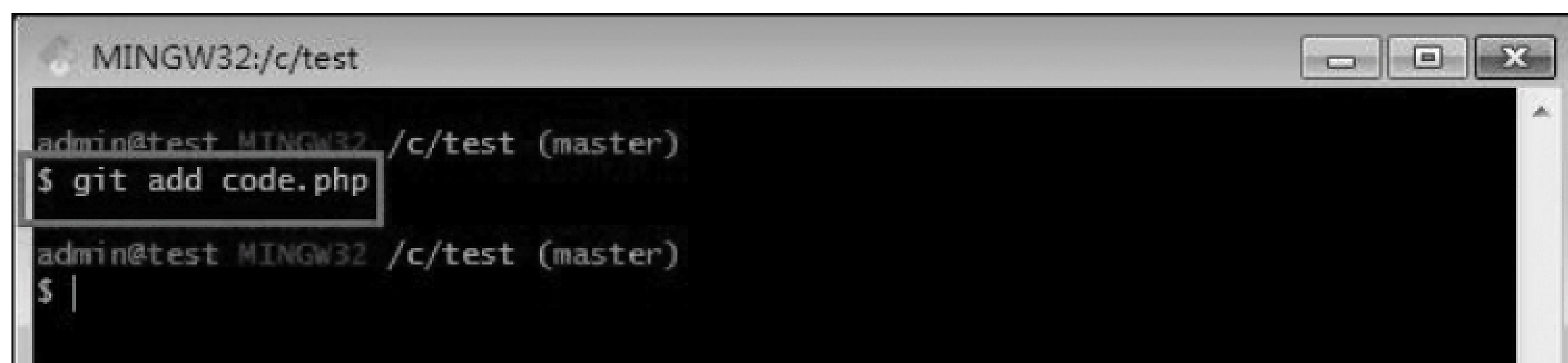


图 4-23　添加至索引库

（5）输入命令“git commit -m 'first commit'”，将索引库中的 code.php 文件提交到本地仓库中，如图 4-24 所示。

```
MINGW32:/c/test
admin@test MINGW32 /c/test (master)
$ git add code.php

admin@test MINGW32 /c/test (master)
$ git commit -m 'first commit'
[master (root-commit) ae850e6] first commit
 1 file changed, 23 insertions(+)
 create mode 100644 code.php

admin@test MINGW32 /c/test (master)
$
```

图 4-24　提交文件

（6）输入命令“git remote add origin http://172.16.1.20:1000/r/test.git”，添加远程仓库，如图 4-25 所示。

```
MINGW32:/c/test
admin@test MINGW32 /c/test (master)
$ git add code.php

admin@test MINGW32 /c/test (master)
$ git commit -m 'first commit'
[master (root-commit) ae850e6] first commit
 1 file changed, 23 insertions(+)
 create mode 100644 code.php

admin@test MINGW32 /c/test (master)
$ git remote add origin http://172.16.1.20:1000/r/test.git

admin@test MINGW32 /c/test (master)
$ |
```

图 4-25　添加远程仓库

（7）输入命令“git push -u origin master”，将本地仓库的内容提交到远程仓库 master 分支下。输入完毕后，输入远程仓库用户名 admin，密码 admin，单击 OK 按钮，如图 4-26 所示。

```
MINGW32:/c/test
admin@test MINGW32 /c/test (master)
$ git add code.php

admin@test MINGW32 /c/test (master)
$ git commit -m 'first commit'
[master (root-commit) ae850e6] first commit
 1 file changed, 23 insertions(+)
 create mode 100644 code.php

admin@test MINGW32 /c/test (master)
$ git remote add origin http://172.16.1.20:1000/r/test.git

admin@test MINGW32 /c/test (master)
$ git push -u origin master
Username for 'http://172.16.1.20:1000': admin
```

图 4-26　提交至远程仓库

（8）打开 Chrome 浏览器，在地址栏中输入 Git 服务器地址“http://172.16.1.20:

1000”,进入 Git 服务器管理界面。输入用户名 admin,密码为 admin,单击“登录”按钮,如图 4-27 和图 4-28 所示。

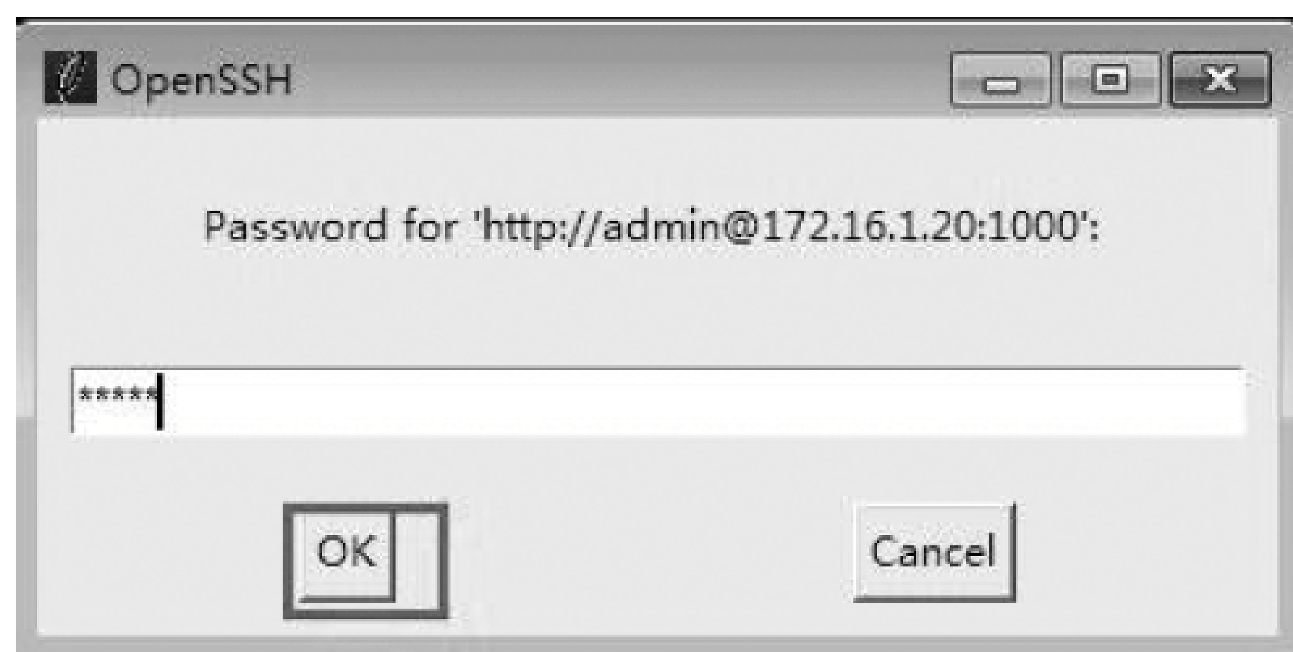


图 4-27　输入密码

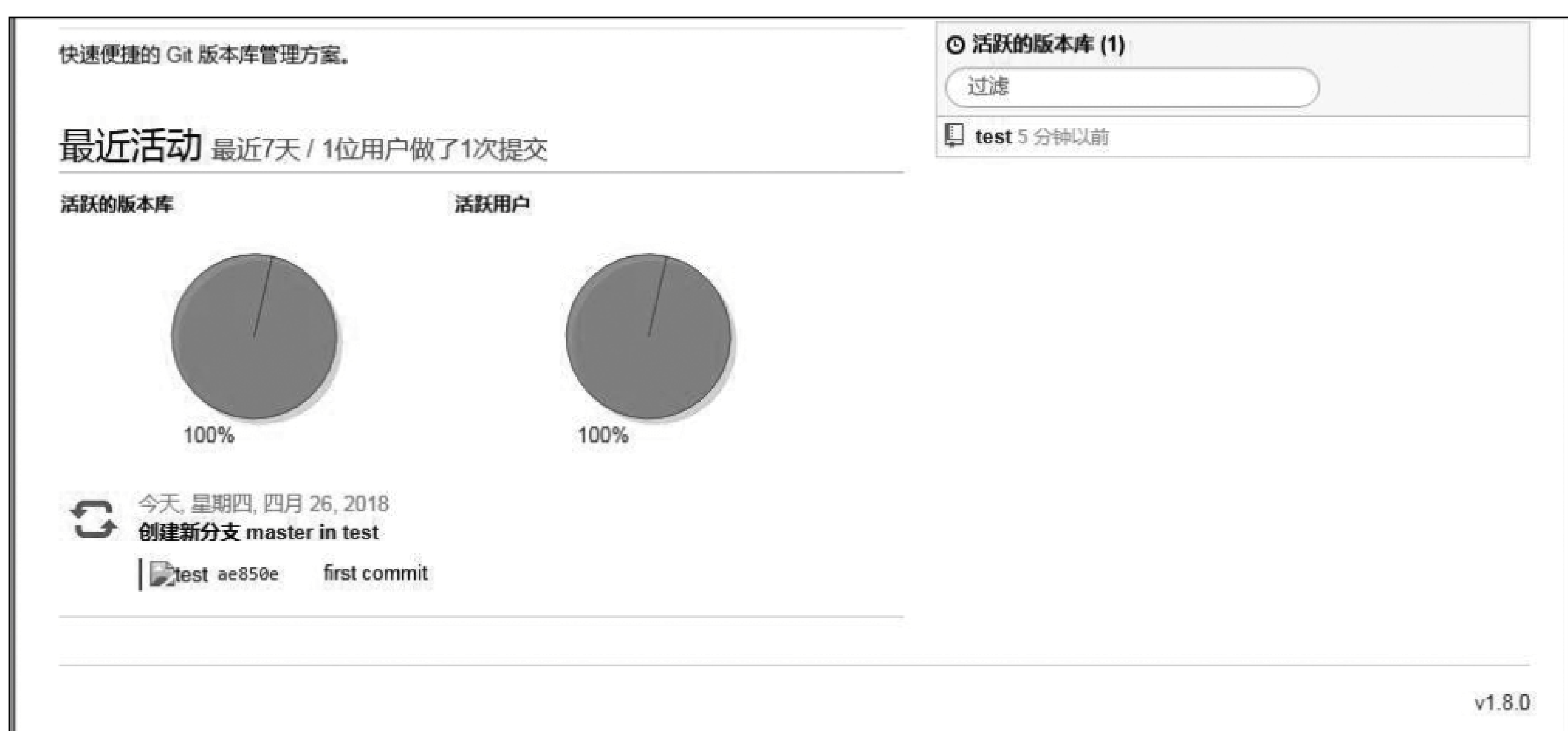


图 4-28　登录 Git

(9) 选择“版本库”命令,如图 4-29 所示。

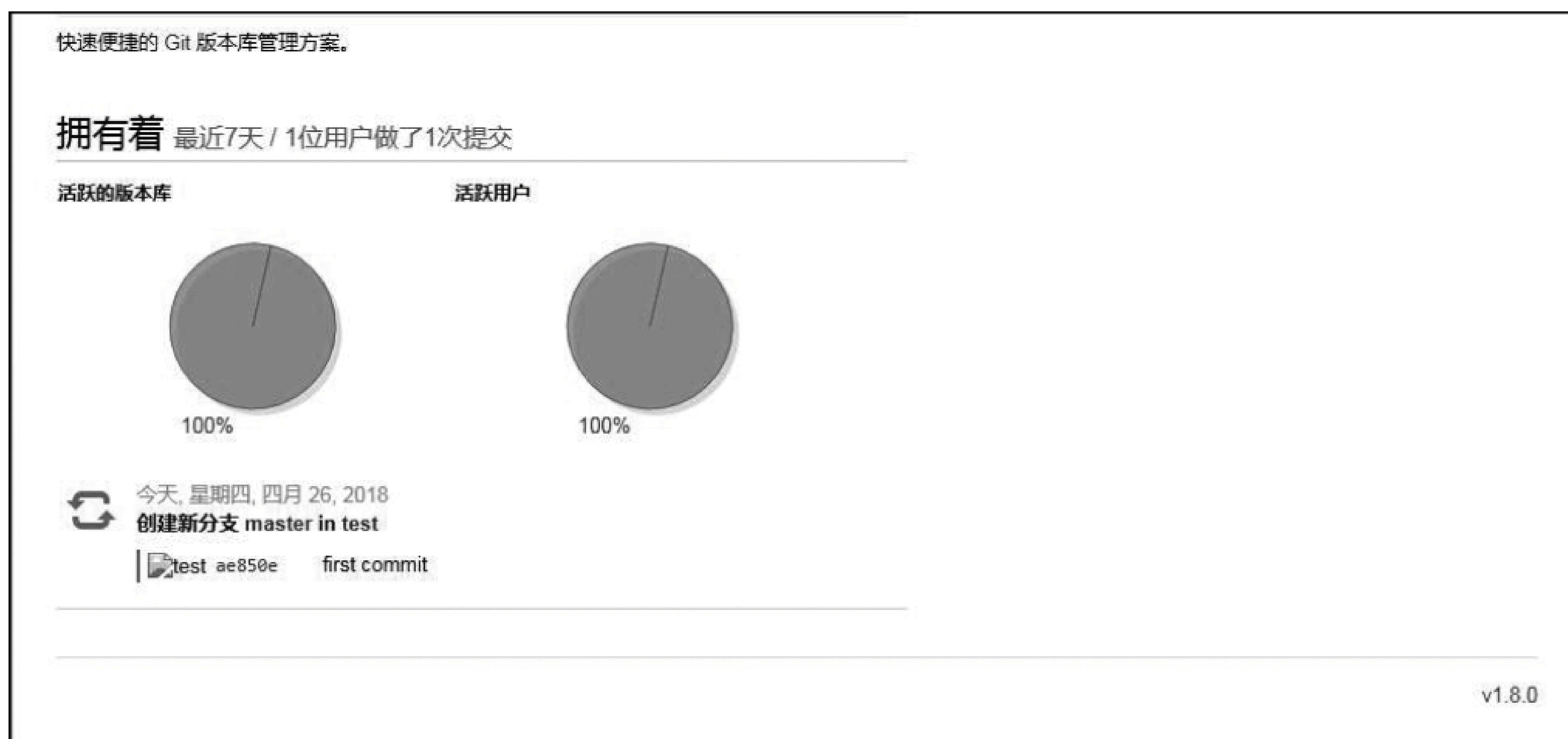


图 4-29　选择“版本库”命令

(10) 选择“测试项目”命令,如图 4-30 所示。



图 4-30　选择“测试项目”命令

(11) 选择 first commit 命令,如图 4-31 所示。



图 4-31　选择 first commit 命令

(12) 可以看到新上传的 code.php 代码文件,如图 4-32 所示。



图 4-32　查看新上传的 code.php

(13) 新建浏览器界面,单击“打开新的标签页”按钮,如图 4-33 所示。

(14) 输入代码卫士的 IP 地址“https://172.16.1.100”,在显示的“您的连接不是私密

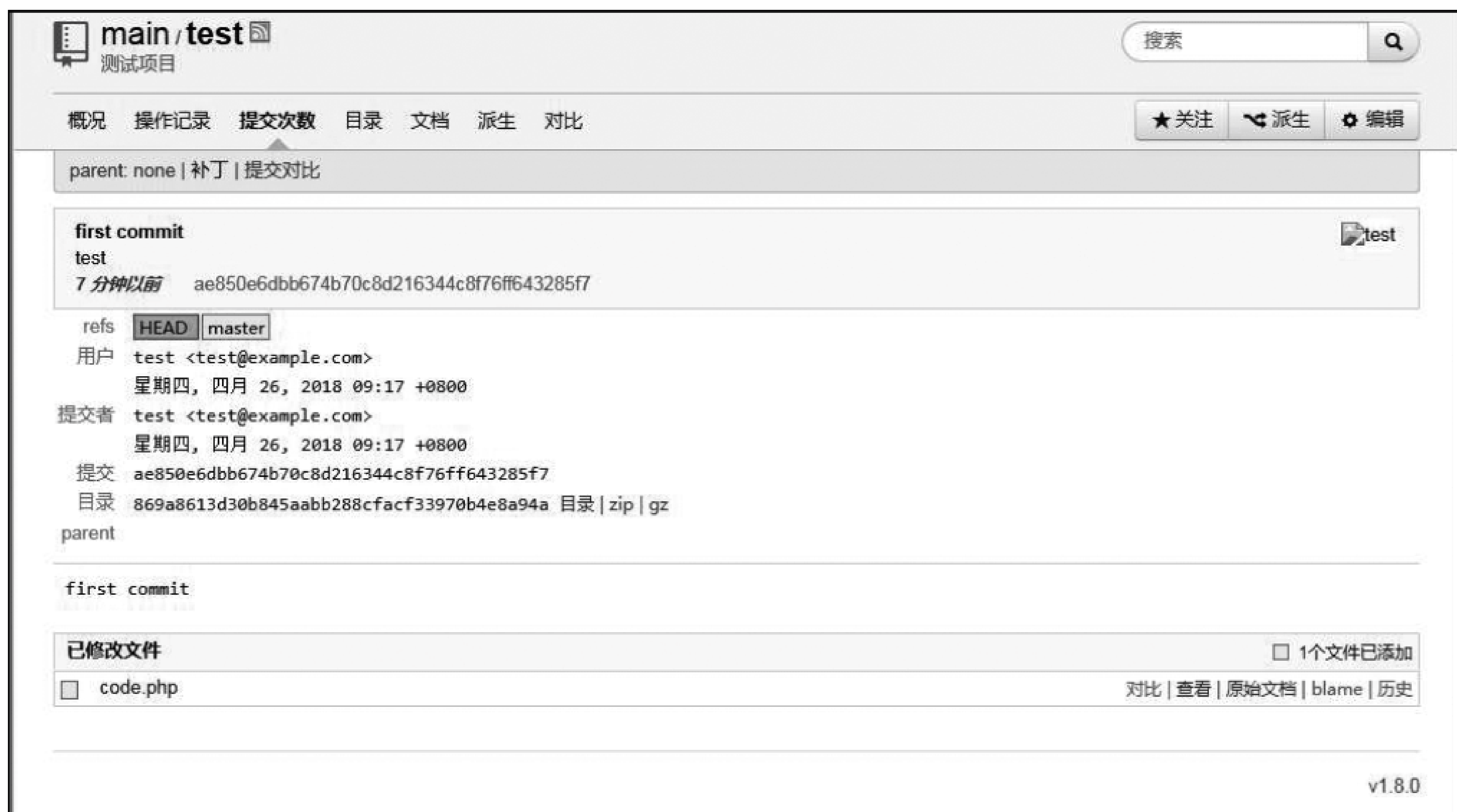


图 4-33　新建浏览器界面

连接”界面中单击“高级”按钮。

(15) 单击“继续前往 172.16.1.100(不安全)”按钮。

(16) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@＃”),单击“登录”按钮。

(17) 选择“项目管理”命令,如图 4-34 所示。



图 4-34　选择“项目管理”命令

(18) 单击“＋新建项目”按钮,如图 4-35 所示。

(19) “项目名称”输入“发起持续检测”,“开发语言”选中 PHP 单选钮,“开始日期”“执行策略”等选择默认,单击“下一步”按钮,如图 4-36 所示。

(20) 单击“添加远程仓库代码”按钮,如图 4-37 所示。

(21) “源代码名称”输入 code,“源代码仓库类型”选中 Git 单选钮,“源代码仓库地



图 4-35　新建项目



图 4-36　发起持续检测



图 4-37　单击“添加远程仓库代码”按钮

址”输入“http://172.16.1.20:1000/r/test.git”,“Git 分支名称”默认为 master,“用户名”为 admin,“密码”为 admin,单击“保存”按钮,如图 4-38 所示。

(22) 单击“下一步”按钮,如图 4-39 所示。

(23) “违禁类别”设置为“全选”,其他保持默认配置,单击“发起检测”按钮,如图 4-40

图 4-38　单击“保存”按钮

图 4-39　单击“下一步”按钮

图 4-40　单击“发起检测”按钮

所示。

【实验预期】

代码卫士根据设定的执行策略会自动对 Git 服务器的代码定期执行检测任务。

【实验结果】

(1) 代码卫士执行检测任务的开始时间由实验设定的开始日期和执行策略决定，检测结束时间根据被测对象不同而不同。本次实验采用系统默认的当前日期作为开始日期，待检测代码的规模也不大，根据计划发起时间和创建时间等待几分钟后，单击“刷新”按钮刷新界面，如图 4-41 所示。

图 4-41　刷新界面

(2) 如果持续检测执行次数仍旧是 0，等待一会儿再次刷新界面。当持续检测执行次数为 1 时，表示系统已经检测过代码，选择左侧的“发起持续检测”命令，如图 4-42 所示。

图 4-42　选择“发起持续检测”命令

(3) 单击 Code 左侧的“＋”按钮，如图 4-43 所示。

图 4-43　单击“＋”按钮

(4) 单击“检测批次”下方的“＋”按钮，如图 4-44 所示。

(5) 可以看到代码卫士已经将 Git 仓库中的代码检测完毕，危险等级为高，如图 4-45

所示。



图 4-44　继续单击“＋”按钮



图 4-45　检测结果(4.2.2)

【实验思考】

(1) 如何设置执行策略为每两周的周日检测?

(2) 如何只检测代码注入缺陷?

4.2.3　代码安全保障系统发起项目里程碑检测实验

【实验目的】

通过使用代码卫士对 Git 服务器中的代码按照周期性进行持续性缺陷检测,期间再进行一次里程碑检测,使学生掌握使用代码卫士发起里程碑检测的方法。

【知识点】

发起里程碑检测任务。

【场景描述】

A 公司研发部编写的代码使用 Git 服务管理，使用代码安全保障系统对提交的代码按照周期进行持续性缺陷检测，其间想发起一次临时的检测任务，检测刚批量提交的代码是否有安全类缺陷。请帮助小王完成里程碑检测任务的发起。

【实验原理】

项目管理可以定义检测任务的执行频率和执行周期，支持定期执行检测任务，每执行一次，称为一个检测批次。

在任务执行过程中，可以发起里程碑任务，即单次特定时间执行的检测任务。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。
- Git 服务器：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-46 所示。

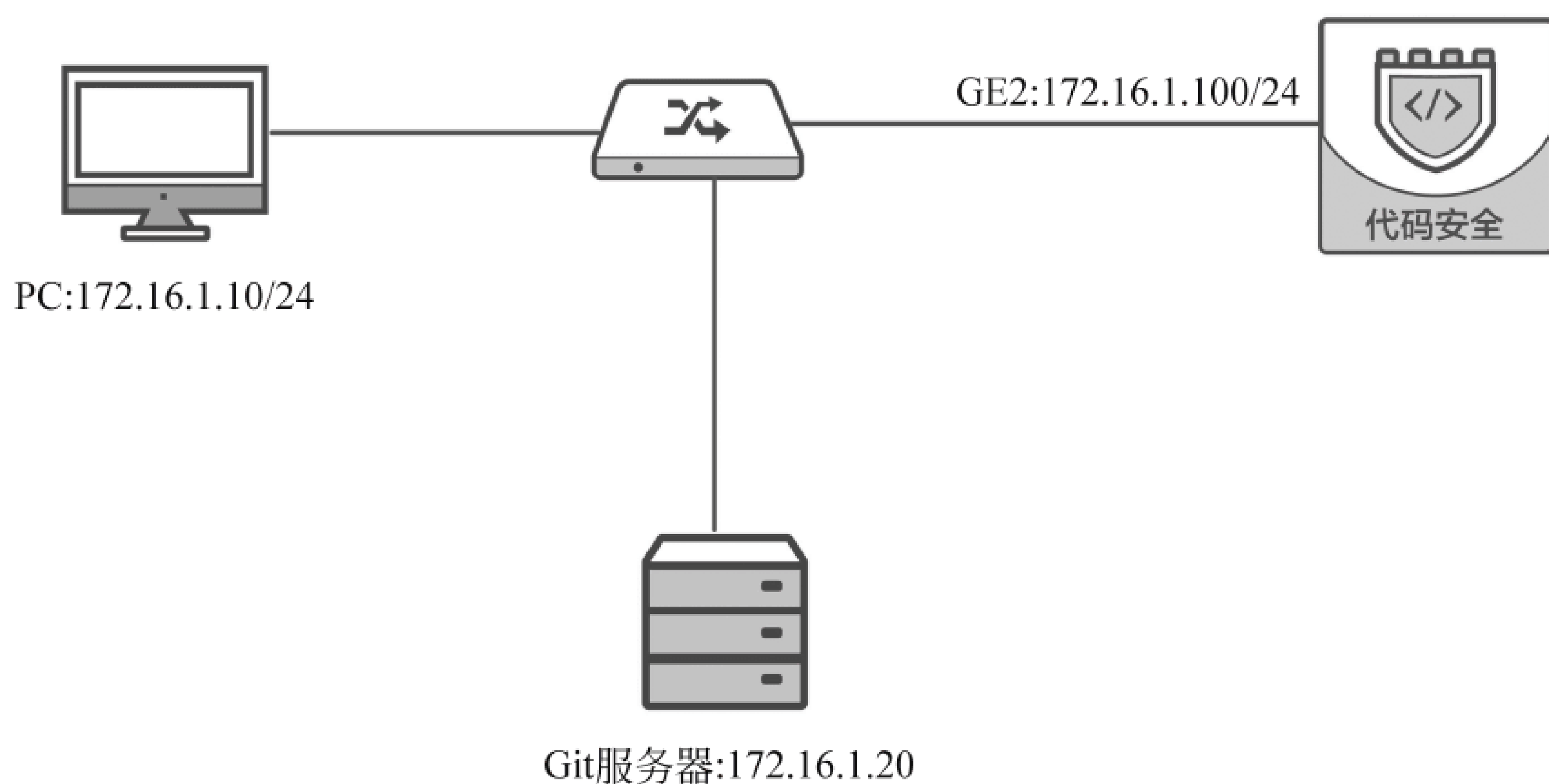


图 4-46　代码安全保障系统发起项目里程碑检测实验拓扑图

【实验思路】

(1) 将代码上传至远程 Git 仓库。

(2) 发起持续检测任务。

(3) 发起里程碑检测任务。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧 PC 主机终端虚拟机，如需登录密码，输入 123456。

(2) 进入“C:\code”目录下，该目录下存放有实验需要用到的 code.php 和 code-fix.php 代码文件。

(3) 在“C:\code”目录下右击，在弹出的快捷菜单中选择“Git Bash Here”命令，进入Git命令行界面。

(4) 输入“git add code.php”，将code.php文件的信息添加到索引库中。

(5) 输入“git commit -m 'commit code.php'”，将索引库中的code.php文件提交到本地仓库中。

(6) 输入“git remote add origin http://172.16.1.20:1000/r/test.git”，添加远程仓库。

(7) 输入“git push -u origin master”，将本地仓库的内容提交到远程仓库master分支下。输入完毕后，输入远程仓库用户名admin和密码admin，单击OK按钮。

(8) 打开Chrome浏览器，输入代码卫士的IP地址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

(9) 单击“继续前往172.16.1.100(不安全)”按钮。

(10) 进入代码卫士登录界面。输入用户名和密码(默认用户名为admin，密码为“admin123!@＃”)，单击“登录”按钮。

(11) 进入代码卫士后，选择“项目管理”命令。

(12) 单击“＋新建项目”按钮。

(13) “项目名称”输入“里程碑检测”，“开发语言”选中PHP单选钮，“开始日期”“执行策略”等保持默认，单击“下一步”按钮。

(14) 选择“添加远程仓库代码”命令。

(15) “源代码名称”输入“里程碑检测code”，“源代码仓库类型”设置为Git，“源代码仓库地址”设置为“http://172.16.1.20:1000/r/test.git”，“Git分支名称”设置为master，用户名和密码均为admin，单击“保存”按钮。

(16) 单击“下一步”按钮。

(17) “违禁类别”设置为全选，其他保持默认，单击“发起检测”按钮。

(18) 根据创建时间和计划发起时间，可以知道大概距离代码检测还有多长时间，如图4-47所示。



图4-47　里程碑检测

(19) 等待一段时间后(根据上一步算出)，选择“里程碑检测”命令，如图4-48所示。

(20) 单击“里程碑检测code”右侧的“＋”按钮，如图4-49所示。

(21) 单击“检测批次”下面的“＋”按钮，如图4-50所示。

(22) 可以看到检测结果，如图4-51所示。

(23) 最小化浏览器窗口，返回Git客户端，输入命令“git pull origin master”，将远程



图 4-48　选择“里程碑检测”命令



图 4-49　单击“里程碑检测 code”左侧的“＋”按钮



图 4-50　选择检测批次 1



图 4-51　检测结果(4.2.3)

仓库代码拉取到本地，如图 4-52 所示。

```
MINGW32:/c/code
admin@test MINGW32 /c/code (master)
$ git remote add origin http://172.16.1.20:1000/r/test.git

admin@test MINGW32 /c/code (master)
$ git push -u origin master
Username for 'http://172.16.1.20:1000': admin
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 579 bytes | 579.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Updating references: 100% (1/1)
To http://172.16.1.20:1000/r/test.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

admin@test MINGW32 /c/code (master)
$ git pull origin master
From http://172.16.1.20:1000/r/test
 * branch            master     -> FETCH_HEAD
Already up to date.

admin@test MINGW32 /c/code (master)
$
```

图 4-52　将远程仓库代码拉取到本地

（24）输入命令“git rm code.php”，删除 code.php 文件，如图 4-53 所示。

```
MINGW32:/c/code
admin@test MINGW32 /c/code (master)
$ git push -u origin master
Username for 'http://172.16.1.20:1000': admin
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 579 bytes | 579.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Updating references: 100% (1/1)
To http://172.16.1.20:1000/r/test.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

admin@test MINGW32 /c/code (master)
$ git pull origin master
From http://172.16.1.20:1000/r/test
 * branch            master     -> FETCH_HEAD
Already up to date.

admin@test MINGW32 /c/code (master)
$ git rm code.php
rm 'code.php'

admin@test MINGW32 /c/code (master)
$
```

图 4-53　删除文件

（25）输入命令“git add code-fix.php”，将 code-fix.php 文件添加到索引库，如图 4-54 所示。

（26）输入命令“git commit -m 'remove code.php，add code-fix.php'”，将本次所有操作提交到本地仓库，如图 4-55 所示。

（27）输入“git push origin master”，将本地仓库的内容提交到远程仓库 master 分支下。输入完毕后，输入远程仓库用户名 admin 和密码 admin，单击 OK 按钮。

（28）返回浏览器代码卫士界面，单击“＋发起里程碑任务”按钮，如图 4-56 所示。

（29）保持默认配置，单击“保存”按钮，如图 4-57 所示。

【实验预期】

代码卫士显示里程碑检测结果。

```
MINGW32:/c/code
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 579 bytes | 579.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Updating references: 100% (1/1)
To http://172.16.1.20:1000/r/test.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

admin@test MINGW32 /c/code (master)
$ git pull origin master
From http://172.16.1.20:1000/r/test
 * branch            master     -> FETCH_HEAD
Already up to date.

admin@test MINGW32 /c/code (master)
$ git rm code.php
rm 'code.php'

admin@test MINGW32 /c/code (master)
$ git add code-fix.php

admin@test MINGW32 /c/code (master)
$
```

图 4-54　添加文件至索引库

```
MINGW32:/c/code
Branch 'master' set up to track remote branch 'master' from 'origin'.

admin@test MINGW32 /c/code (master)
$ git pull origin master
From http://172.16.1.20:1000/r/test
 * branch            master     -> FETCH_HEAD
Already up to date.

admin@test MINGW32 /c/code (master)
$ git rm code.php
rm 'code.php'

admin@test MINGW32 /c/code (master)
$ git add code-fix.php

admin@test MINGW32 /c/code (master)
$ git commit -m 'remove code.php,add code-fix.php'
[master 707e7fd] remove code.php,add code-fix.php
 2 files changed, 32 insertions(+), 23 deletions(-)
 create mode 100644 code-fix.php
 delete mode 100644 code.php

admin@test MINGW32 /c/code (master)
$
```

代码卫士 codesafe　首页　快速检测　项目管理　报告管理　统计分析　系统管理　当前用户：admin　资源下载

任务列表　差距分析　+ 发起里程碑任务

当前项目:里程碑检测　搜索

源代码名称	开发语言	最近批次发起时间	执行策略	持续检测执行次数	删除选中项
里程碑检测code	PHP	2018-04-26 17:11:00	每1周的周4	1	

检测批次	批次发起时间	任务类型	检测方式
1	2018-04-26 17:11:00	持续检测	缺陷检测

检测方式	检测开始时间	检测结束时间	检测状态	检测结果	等级分布	操作
缺陷检测	2018-04-26 17:11:00	2018-04-26 17:11:26	检测完成	1		

图 4-56　发起里程碑任务

基本信息
源代码名称： 里程碑检测code
缺陷检测模板： 缺陷检测模板[PHP]
开始日期： 2018-04-26
开始时间： 17 : 23
保存 关闭

图 4-57　单击“保存”按钮

【实验结果】

(1) 等待 5min 左右(本次实验采用系统默认的开始日期和开始时间，即当前发起检测任务日期，待检测的代码规模很小，因此所需等待时间不超过 5min。在实际工作中等待时间依据开始日期和开始时间的不同而不同，且要确保实验主机的时间与代码卫士时间一致)，单击“刷新”按钮，刷新界面。

(2) 如果持续检测执行次数仍旧是 0，等待一会儿再次刷新界面。当持续检测执行次数为 1 时，表示系统已经检测过代码，单击“里程碑检测 code”左侧的“＋”按钮，如图 4-58 所示。



图 4-58　展开“里程碑检测 code”

(3) 可以发现第一条检测记录的“任务类型”为“里程碑检测”，单击“Mark-1”左侧的“＋”按钮，如图 4-59 所示。



图 4-59　单击“Mark-1”左侧的“＋”按钮

(4) 可以看到,修改后的 code-fix.php 已经没有缺陷了,如图 4-60 所示。



图 4-60　检测结果(4.2.3)

【实验思考】

(1) 发起里程碑检测有什么意义?

(2) 如果只想检测跨站脚本缺陷,该如何设置?

4.3　发起 Java&Python 检测任务

4.3.1　Java 语言缺陷检测任务实验

【实验目的】

通过使用代码安全保障系统来检测使用 SVN 服务管理的代码模块是否存在安全类缺陷,并对发现的安全类缺陷进行有针对性的修复,确保所编写的代码中不存在安全类缺陷。

【知识点】

Java 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写的代码模块使用 SVN 服务管理,需要使用代码安全保障系统对编写的代码进行安全类缺陷检测,通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成远程代码的缺陷检测任务。

【实验原理】

各检测引擎接收来自集成编译器模块的编译信息,从缺陷知识库模块获取相应的检测策略进行检测,并将检测的结果反馈给安全管理平台。源代码安全管理平台可以通过

外部接口模块从 SVN、Git 等代码管理系统获取软件源代码，然后交给缺陷检测引擎来检查源代码安全缺陷。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-61 所示。

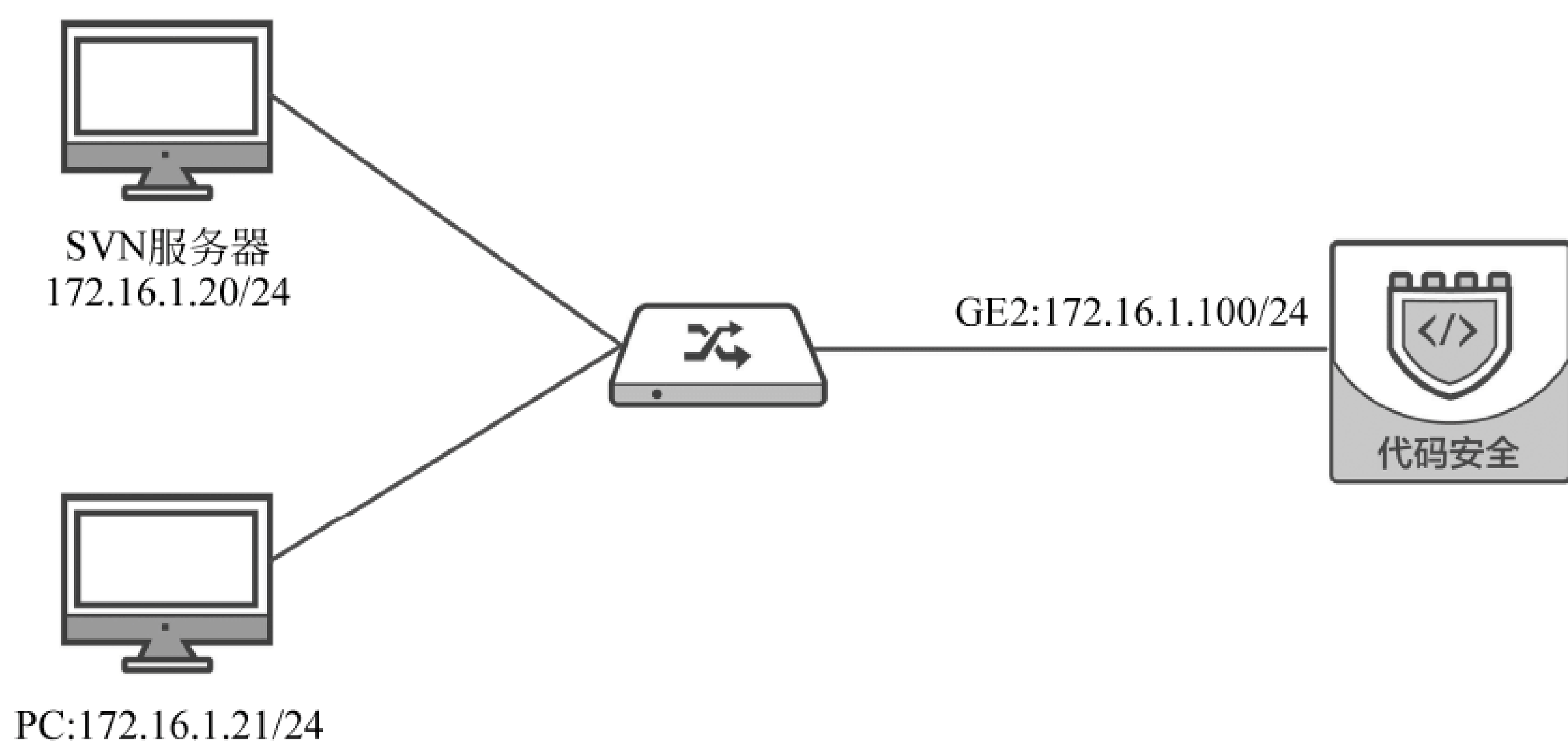


图 4-61　Java 语言缺陷检测任务实验拓扑图

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 查看效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左上方 SVN 主服务器，如需登录密码，输入 123456。

(2) 运行"C:\svnmanager\"目录下的 VisualSVN Server Manager 程序，如图 4-62 所示。



图 4-62　运行 SVN 服务器

(3) 选择左侧框中的 Repositories 命令,展开目录,如图 4-63 所示。

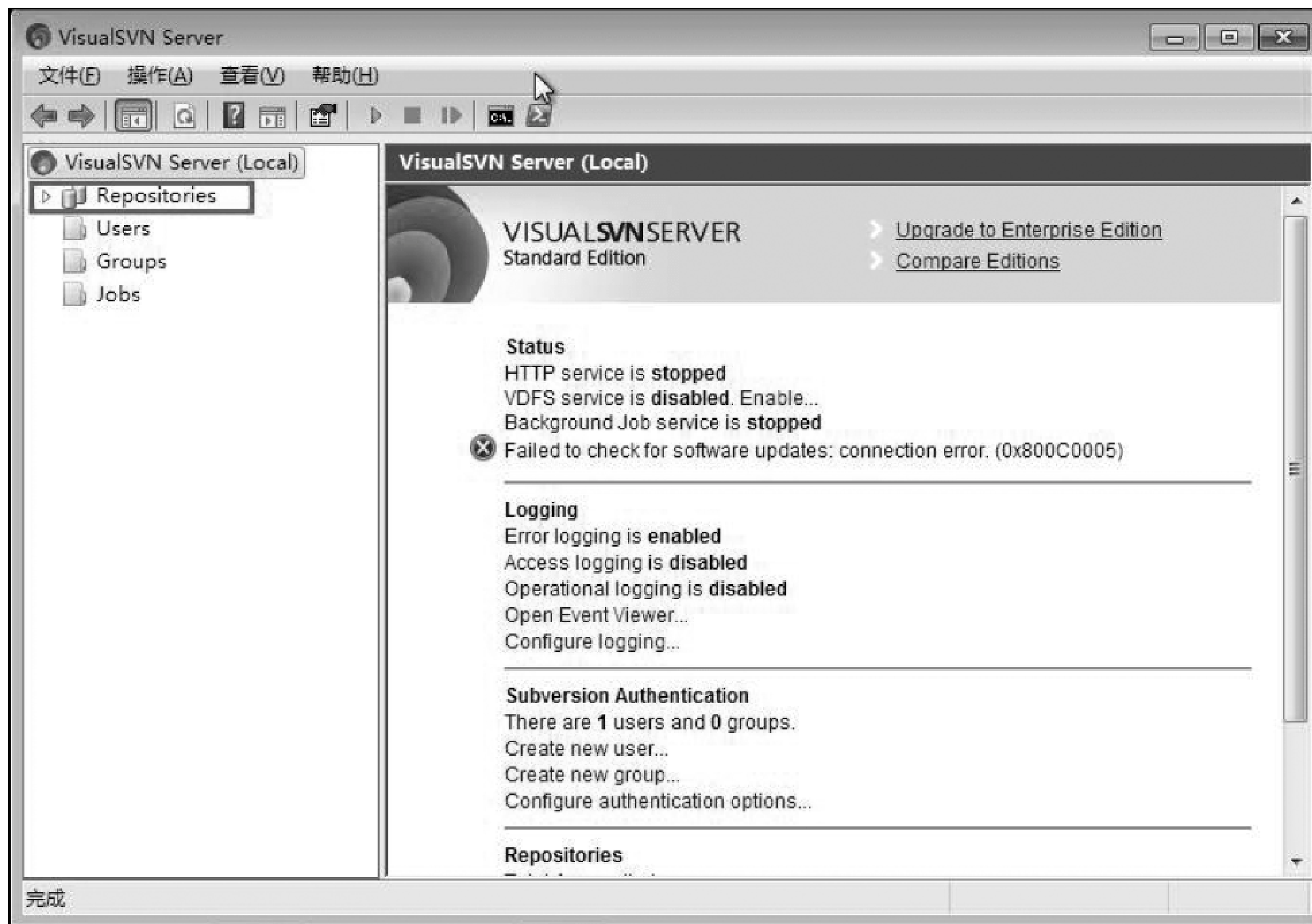


图 4-63　部分代码展示(4.3.1)

(4) 可以看到 SVN 仓库中的 Java 工程文件,如图 4-64 所示。

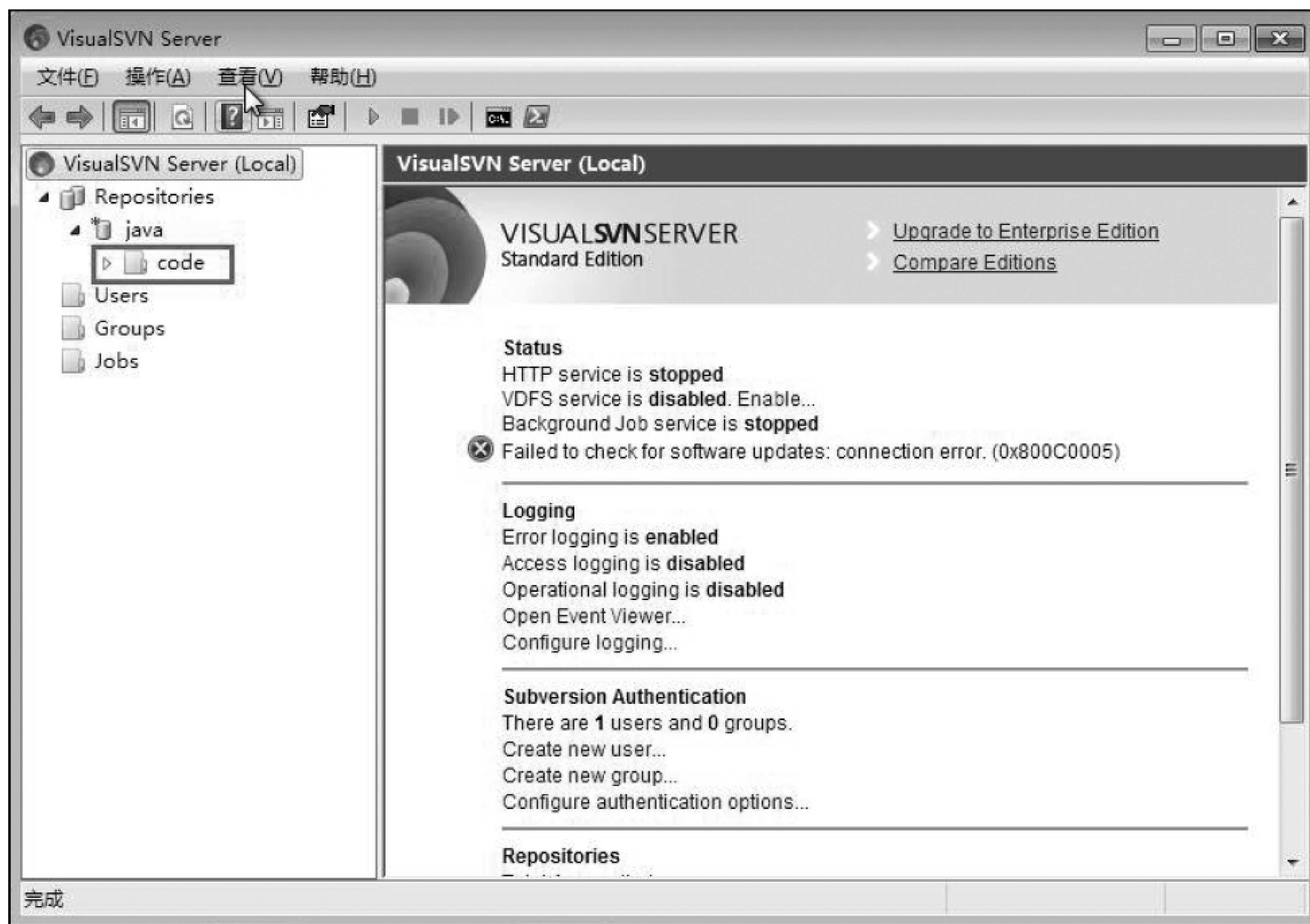


图 4-64　项目工程

（5）单击“启动 SVN 服务器”按钮，如图 4-65 所示。

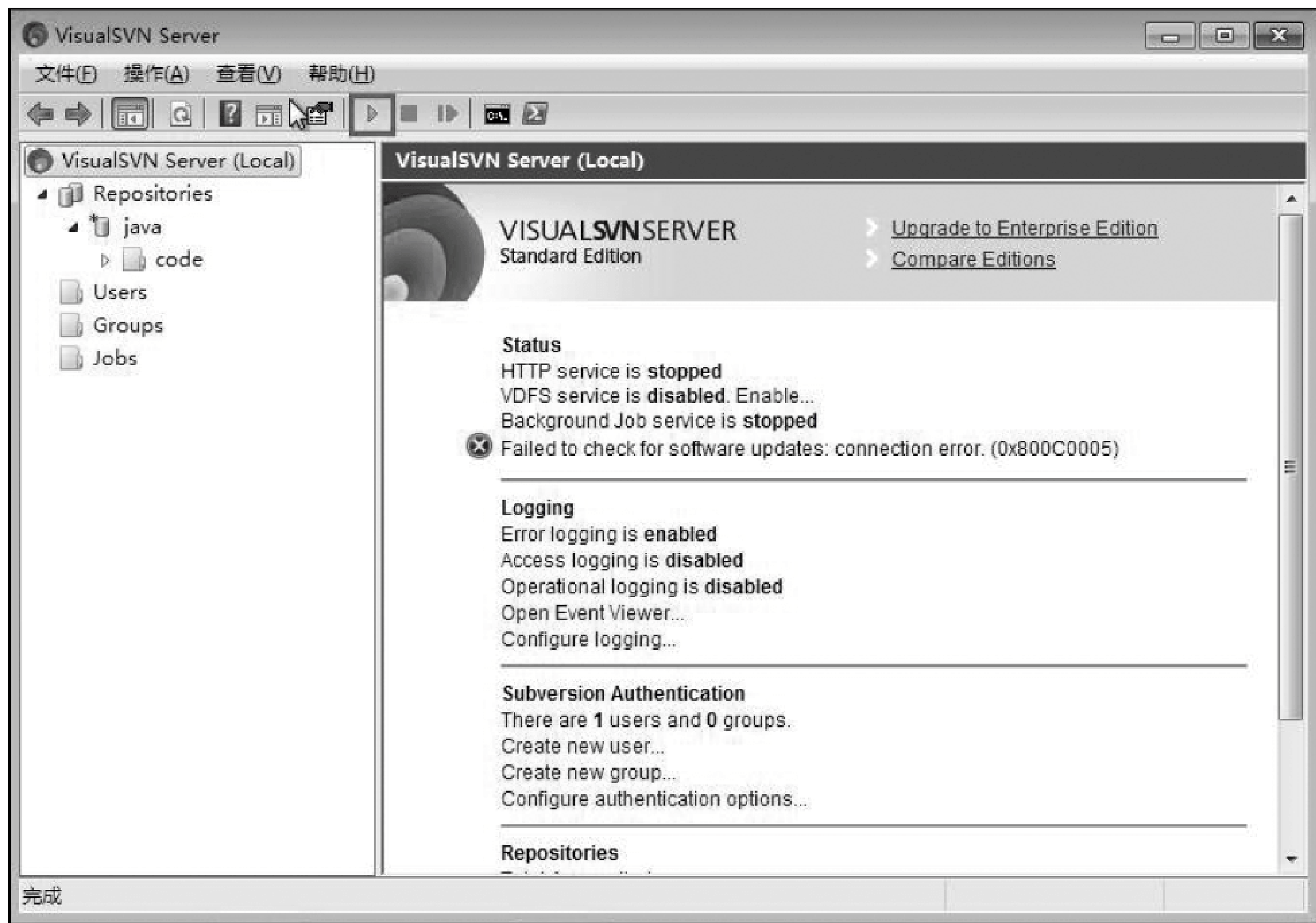


图 4-65　启动 SVN

（6）退出登录，进入实验平台对应的实验拓扑，登录左下方的 PC 虚拟机，如需登录密码，输入 123456。

（7）打开 Chrome 浏览器，进入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

（8）在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

（9）跳转到登录界面，输入用户名与密码（默认用户名为 admin，密码为“admin123!@#”），单击“登录”按钮。

（10）进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

（11）在配置信息界面中，“任务名称”输入“Java 语言缺陷任务”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，“是 J2EE 工程”选中“否”单选钮，“源码来源”选中 Svn 单选钮，“Svn 地址”输入 https://172.16.1.20/svn/java，“Svn 用户名”输入 test，“Svn 密码”输入 123456，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮，如图 4-66 所示。

【实验预期】

可以直接通过 SVN 代码库发起代码检测。

【实验结果】

发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。检测完成，如图 4-67 所示。

图 4-66　发起检测

图 4-67　检测完成(4.3.1)

【实验思考】

(1) 源码来源选择 SVN,发起 Java 语言合规检测任务,查看结果。

(2) 源码来源选择 SVN,发起 Java 语言溯源检测任务,查看结果。

4.3.2　Java 语言自定义检测模板缺陷检测任务实验

【实验目的】

通过使用代码安全保障系统来检测使用 SVN 服务管理的代码模块是否存在自定义检测模板中的缺陷,并对发现的此类缺陷进行针对性的修复,确保所编写的代码中不存在此类缺陷。

【知识点】

Java 代码缺陷检测。

【场景描述】

A 公司研发部工程师小王编写的代码使用 SVN 服务管理，需要使用代码安全保障系统对编写的代码进行安全类缺陷检测，只检测研发部关注的缺陷分类，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成自定义检测模板任务的发起。

【实验原理】

缺陷知识库模块是代码卫士用于存储源代码检测策略的数据库，由缺陷检测规则库、合规检测规则库、开源组件检测规则库及定制化规则库 4 部分组成。

可检测的缺陷种类包括缓冲区溢出、SQL 注入、跨站脚本、代码质量、危险函数等 13 个大类，600 多个小类；可用于检测的开源组件规则有 60 万条。源代码安全管理平台可以通过外部接口模块从 SVN、Git 等代码管理系统获取软件源代码，然后交给缺陷规则库模块来检查源代码安全缺陷。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-68 所示。

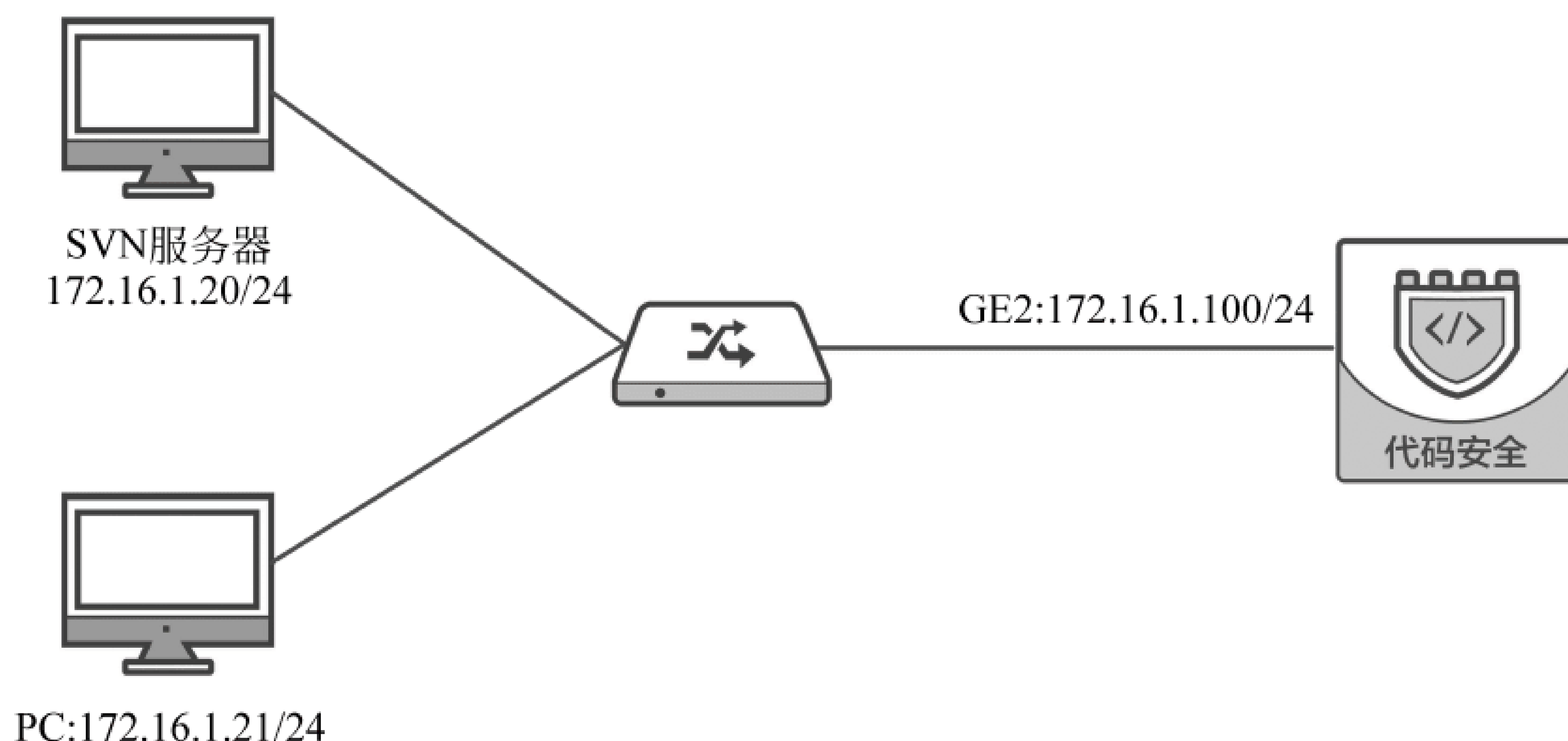


图 4-68　Java 语言自定义检测模板缺陷检测任务实验拓扑图

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 查看效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左上方的 SVN 服务器，如需登录密码，输入 123456。

(2) 运行“C:\svnmanager\”目录下的 VisualSVN Server Manager 程序。

(3) 单击左侧框中的 Repositories 按钮，展开目录。

(4) 可以看到 SVN 仓库中的 Java 工程文件。

(5) 单击“启动 SVN 服务器”按钮。

(6) 退出登录，进入实验平台对应的实验拓扑，登录左下方的 PC 虚拟机，如需登录密码，输入 123456。

(7) 打开 Chrome 浏览器，进入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(8) 在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

(9) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(10) 进入代码卫士后，选择“系统管理”命令。

(11) 选择“检测模板管理”命令。

(12) 在“缺陷检测”目录下，单击“＋添加模板”按钮，如图 4-69 所示。



图 4-69　发起添加模板

(13) 在配置信息界面中，“模板名称”输入“开发部的缺陷检测 java”，“开发语言”选中 Java 单选钮，“缺陷分类”设置为“资源管理”，其他保持默认配置，单击“保存”按钮，如图 4-70 所示。

(14) 选择“快速检测”→“＋发起快速检测”命令。

(15) 在配置信息界面中，“任务名称”输入“自定义模板的缺陷检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，“是 J2EE 工程”选中“否”单选钮，“源码来源”选中 Svn 单选钮，“Svn 地址”输入 https://172.16.1.20/svn/java，“Svn 用户名”输入 test，“Svn 密码”输入 123456，“缺陷检测”设置为“开发部的缺陷检测 java”，取消勾选



图 4-70　配置信息

“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

【实验预期】

可以通过自定义模板检测任务。

【实验结果】

发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。检测完成，如图 4-71 所示。



图 4-71　检测完成(4.3.2)

【实验思考】

(1) 自定义模板选择"代码注入"缺陷分类,源码来源选择 SVN,发起 Java 语言缺陷检测任务,查看结果。

(2) 自定义模板选择"代码质量"缺陷分类,源码来源选择 SVN,发起 Java 语言缺陷检测任务,查看结果。

4.3.3　Java 语言基于 maven 构建的检测任务实验

【实验目的】

通过使用代码安全保障系统来检测使用 maven 管理的代码模块是否存在安全类缺陷,并对发现的安全类缺陷进行有针对性的修复,确保所编写的代码中不存在安全类缺陷。

【知识点】

Java 代码缺陷检测。

【场景描述】

A 公司研发部使用 maven 管理项目的构建,使用代码安全保障系统对编写的代码进行安全类缺陷检测,通过 maven 仓库下载依赖。

请完成基于 maven 构建的检测任务。

【实验原理】

各检测引擎接收来自集成编译器模块的编译信息,从缺陷知识库模块获取相应的检测策略进行检测,并将检测的结果反馈给安全管理平台。源代码安全管理平台可以通过外部接口模块从 SVN、Git 等代码管理系统获取软件源代码,然后交给缺陷检测引擎来检查源代码安全缺陷。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-72 所示。

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 查看效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左下方的 PC 虚拟机,如需登录密码,输入 123456。

(2) 查看"C:\code"目录下的 code.zip 文件,如图 4-73 所示。

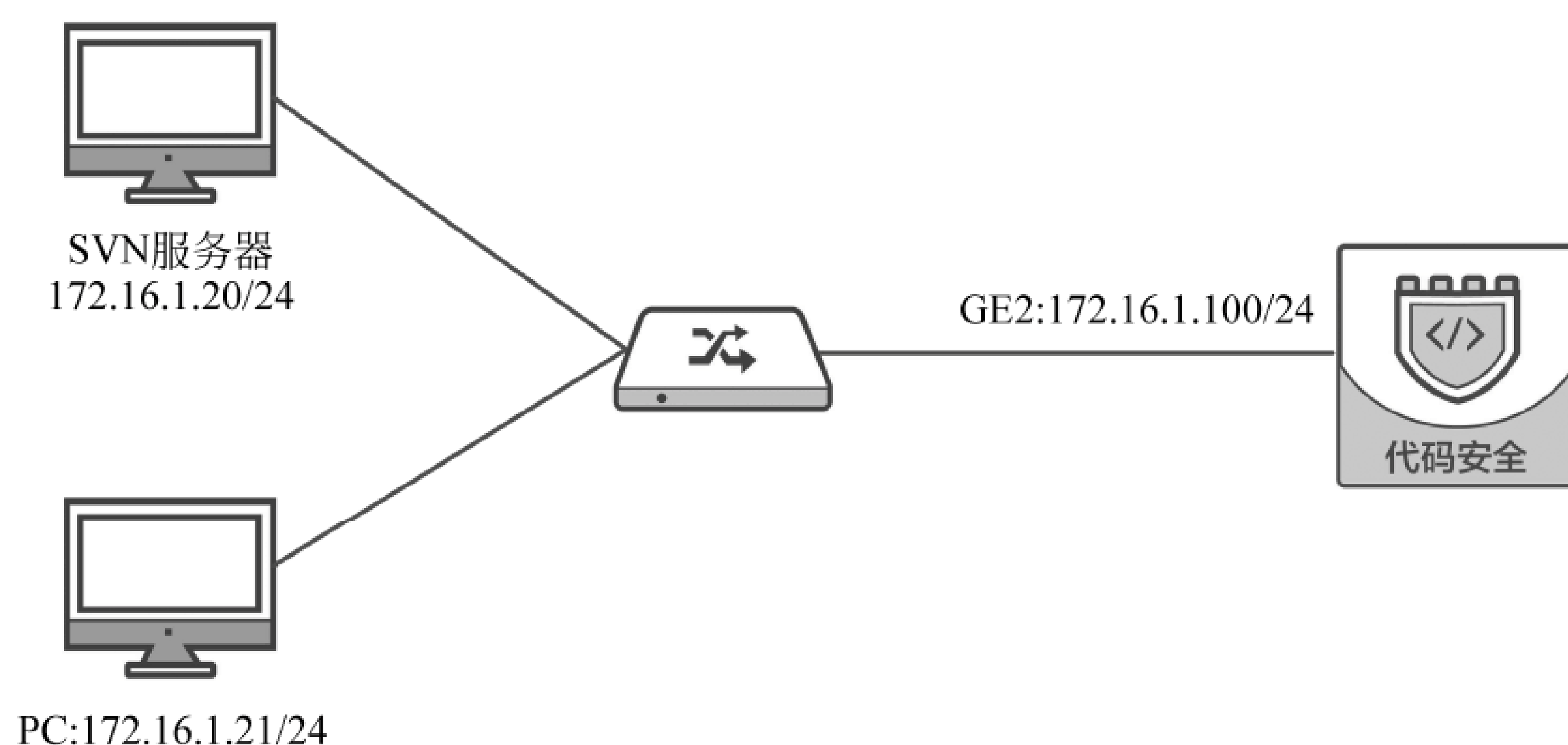


图 4-72 Java 语言基于 maven 构建的检测任务实验拓扑图



图 4-73 查看文件

(3) 打开 Chrome 浏览器,进入代码卫士登录界面网址"https://172.16.1.100"。在显示的"您的连接不是私密连接"界面中,单击"高级"按钮。

(4) 在显示的隐藏折叠信息中,单击"继续前往 172.16.1.100"按钮。

(5) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(6) 进入代码卫士后,选择"系统管理"命令。

(7) 选择"资源管理"下的"maven 管理"命令。

(8) 单击"+添加 maven 代码库"按钮,如图 4-74 所示。



图 4-74 添加 maven 代码库

(9) 在配置信息界面中,“代码库名称”输入 maven,“代码库地址”输入“http://172.16.1.20:8081/nexus/content/groups/public/”,然后单击“保存”按钮,如图 4-75 所示。

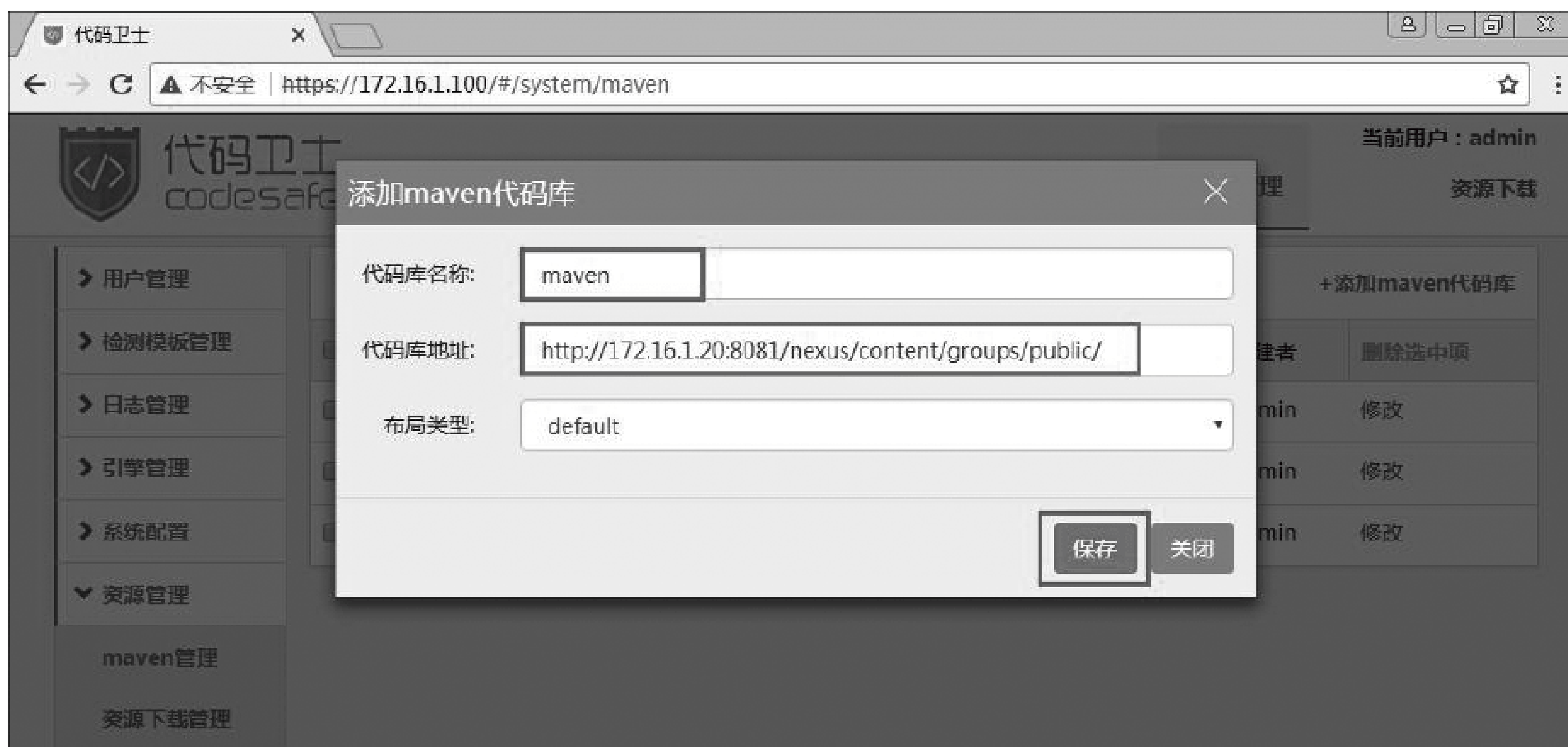


图 4-75　配置信息

(10) 选择“快速检测”→“+发起快速检测”命令。

(11) 在配置信息界面中,“任务名称”输入“Java 语言缺陷任务”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,“是 J2EE 工程”选中“否”单选钮,“源码来源”选中 Svn 单选钮,“Svn 地址”输入“https://172.16.1.20/svn/java”,“Svn 用户名”输入 test,“Svn 密码”输入 123456,取消勾选“合规检测”复选框,取消勾选“溯源检测”复选框,其他保持默认配置,单击“发起检测”按钮。

【实验预期】

可以对基于 maven 构建的任务发起代码检测。

【实验结果】

发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。检测完成,如图 4-76 所示。



图 4-76　检测完成(4.3.3)

【实验思考】

(1) 工程类型选择 maven,发起 Java 语言合规检测任务,查看结果。

（2）工程类型选择 maven，发起 Java 语言溯源检测任务，查看结果。

4.3.4 Java 语言基于 gradle 构建的检测任务实验

【实验目的】

通过使用代码安全保障系统来检测使用 gradle 管理的代码模块是否存在安全类缺陷，并对发现的安全类缺陷进行针对性的修复，确保所编写的代码中不存在安全类缺陷。

【知识点】

Java 代码缺陷检测。

【场景描述】

A 公司研发部使用 gradle 管理项目的构建，使用代码安全保障系统对编写的代码进行安全类缺陷检测，通过 gradle 仓库下载依赖。

请完成基于 gradle 构建的检测任务。

【实验原理】

各检测引擎接收来自集成编译器模块的编译信息，从缺陷知识库模块获取相应的检测策略进行检测，并将检测的结果反馈给安全管理平台。源代码安全管理平台可以通过外部接口模块从 SVN、Git 等代码管理系统获取软件源代码，然后交给缺陷检测引擎来检查源代码安全缺陷。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-77 所示。

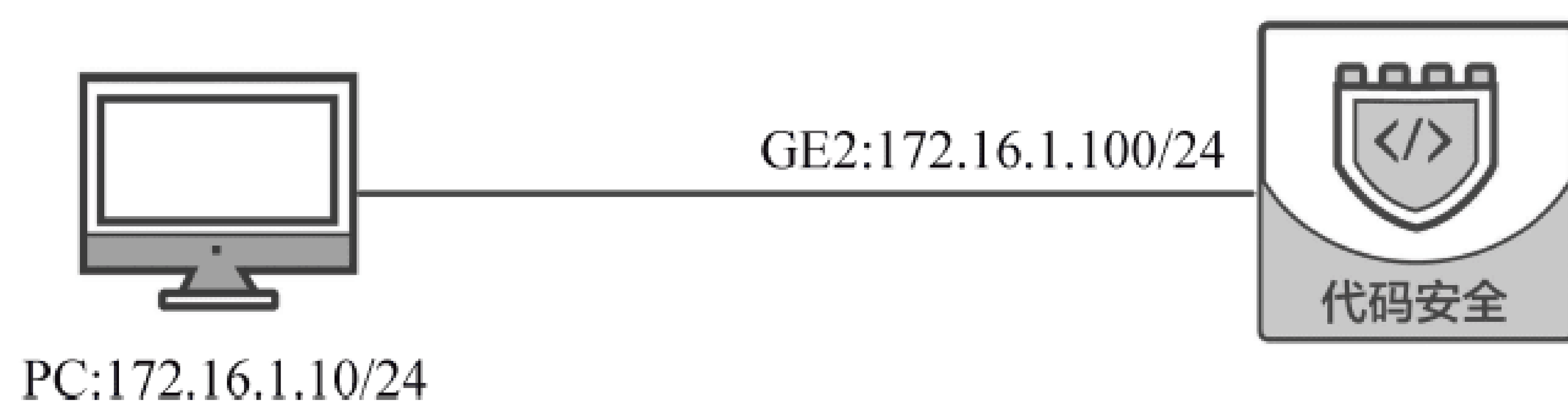


图 4-77　Java 语言基于 gradle 构建的检测任务实验拓扑图

【实验思路】

（1）发起 Java 代码缺陷检测。

（2）查看检测结果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）查看“C:\code\code\”下的项目文件，如图 4-78 所示。



图 4-78　查看项目文件

（3）打开 Chrome 浏览器，进入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

（4）在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

（5）跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

（6）进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

（7）在配置信息界面中，“任务名称”输入“缺陷检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件(此文件是项目文件的压缩包格式，代码卫士平台只能上传“*.zip”文件)，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

【实验预期】

可以对基于 gradle 构建的任务发起代码检测。

【实验结果】

发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。检测完成，如图 4-79 所示。



图 4-79　检测完成(4.3.4)

【实验思考】

(1) 基于 gradle 构建的任务，发起 Java 合规检测任务，查看结果。

(2) 基于 gradle 构建的任务，发起 Java 溯源检测任务，查看结果。

4.3.5 Java 语言合规检测任务实验

【实验目的】

通过使用代码安全保障系统来检测使用 SVN 服务管理的代码模块是否存在违规代码，并对发现的违规代码进行有针对性的修复，确保所编写的代码中不存在违规代码。

【知识点】

Java 代码合规检测。

【场景描述】

A 公司研发部工程师小王编写的代码模块使用 SVN 服务管理，需要使用代码安全保障系统对编写的代码进行违规类缺陷检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成远程代码的合规检测任务。

【实验原理】

合规检测引擎将 CERT C/C++/Java 的安全编码规范进行解析，将每条规范以规则的方式加入检测引擎中去，从而对源代码进行检测。源代码安全管理平台可以通过外部接口模块从 SVN、Git 等代码管理系统获取软件源代码，然后交给合规引擎来检查源代码安全缺陷。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-80 所示。

【实验思路】

(1) 发起 Java 代码合规检测。

(2) 查看效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左上方 SVN 服务器，如需登录密码，输入 123456。

(2) 运行“C:\svnmanager\”目录下的“VisualSVN Server Manager”程序。

(3) 选择左侧框中的 Repositories 命令，展开目录。

(4) 可以看到 Svn 仓库中的 Java 工程文件。

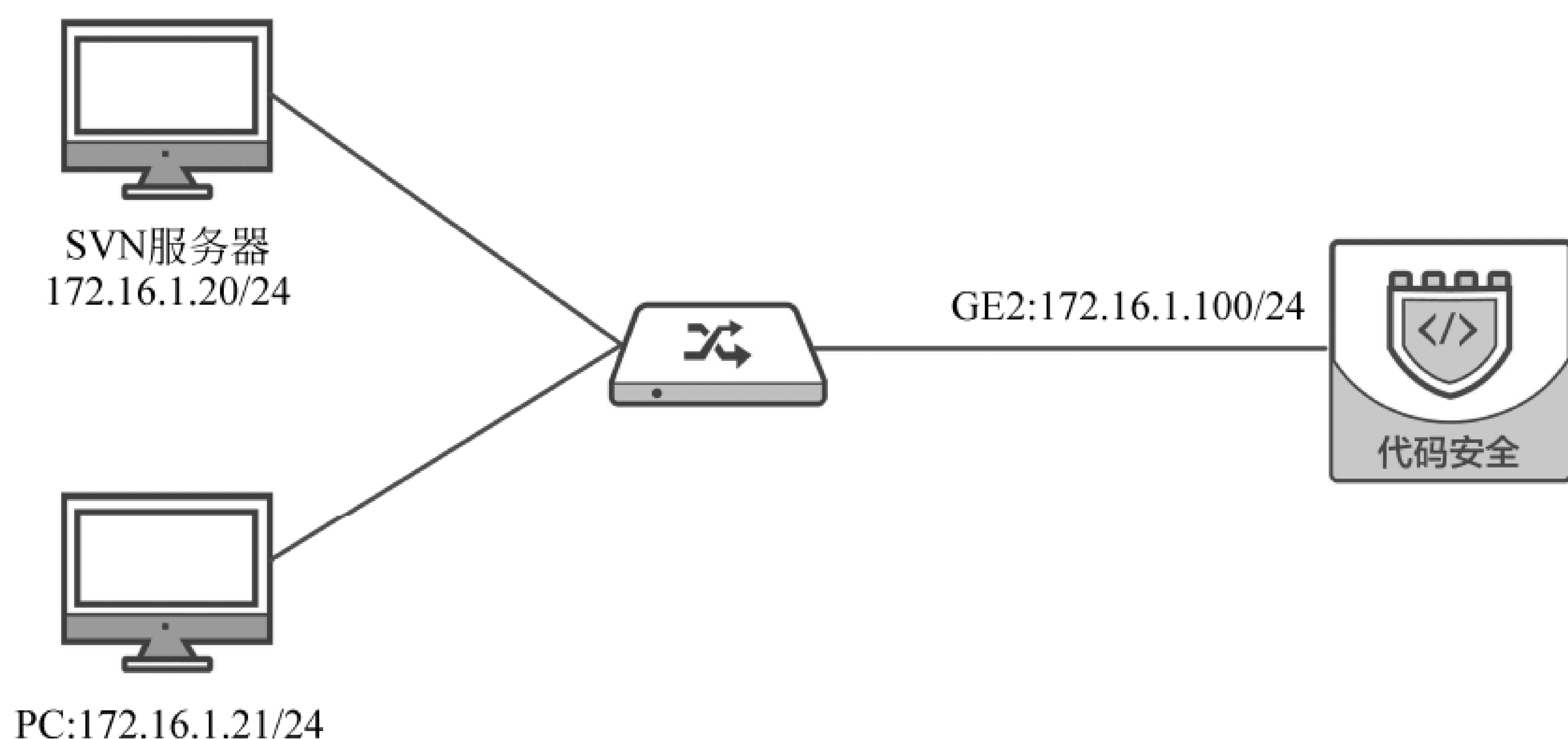


图 4-80　Java 语言合规检测任务实验拓扑图

（5）单击“启动 Svn 服务器”按钮。

（6）退出登录，进入实验平台对应的实验拓扑，登录左下方的 PC 虚拟机，如需登录密码，输入 123456。

（7）打开 Chrome 浏览器，输入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

（8）在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

（9）跳转到登录界面，输入用户名与密码（默认用户名为 admin，密码为“admin123!@#”），单击“登录”按钮。

（10）进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

（11）在配置信息界面中，“任务名称”输入“合规任务检测”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，“是 J2EE 工程”选中“否”单选钮，“源码来源”选中 Svn 单选钮，“Svn 地址”输入“https://172.16.1.20/svn/java”，“Svn 用户名”输入 test，“Svn 密码”输入 123456，取消勾选“缺陷检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

【实验预期】

可以直接通过 SVN 代码库发起代码检测。

【实验结果】

发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息。检测完成，如图 4-81 所示。

【实验思考】

（1）源码来源选择 SVN，发起 Java 语言缺陷检测任务，查看结果。

（2）源码来源选择 SVN，发起 Java 语言溯源检测任务，查看结果。



图 4-81 检测完成(4.3.5)

4.3.6 溯源检测任务实验

【实验目的】

通过使用代码安全保障系统检测上传代码模块是否引用了开源代码模块，是否存在软件使用授权的问题，帮助组织规避开源组件的法律风险。

【知识点】

Java 代码溯源检测。

【场景描述】

A 公司研发部工程师小王以开源代码为基础编写了一个代码模块，需要使用代码安全保障系统对编写的代码进行溯源检测，通过代码安全保障系统的检测结果调整和优化代码模块。

请帮助小王完成代码模块的溯源检测。

【实验原理】

溯源检测模块遍历待检测源代码的组件信息，与缺陷知识库的开源代码检测规则库进行比对，如匹配到具体的开源组件，便将该组件的信息及存在的安全漏洞信息反馈到安全管理平台。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-82 所示。

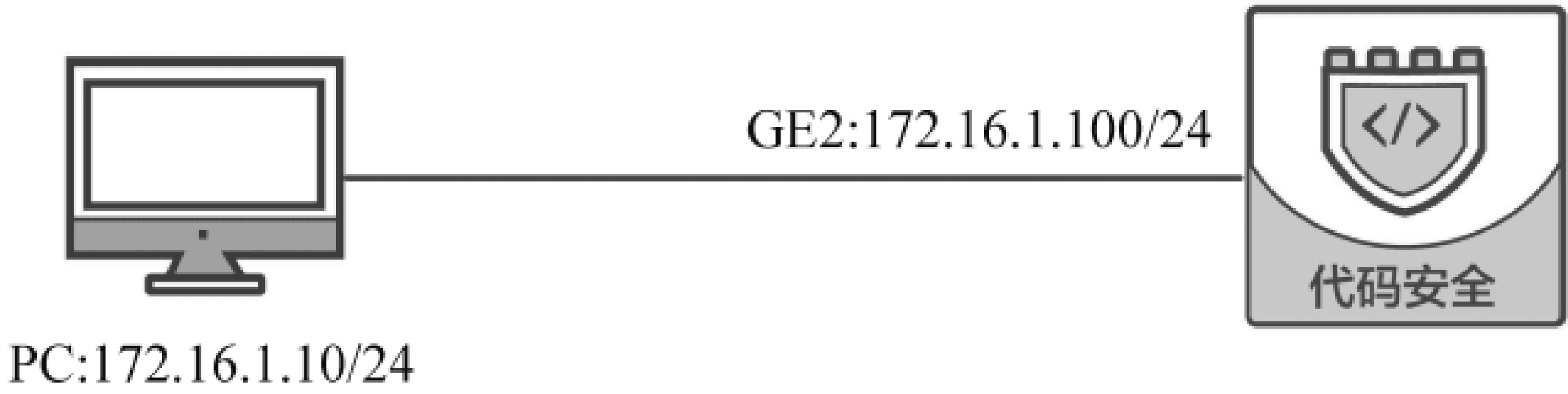


图 4-82 溯源检测任务实验拓扑图

【实验思路】

(1) 发起 Java 代码溯源检测。

(2) 查看检测效果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 打开“C:\code\”目录下的“code.zip”文件,如图 4-83 所示。

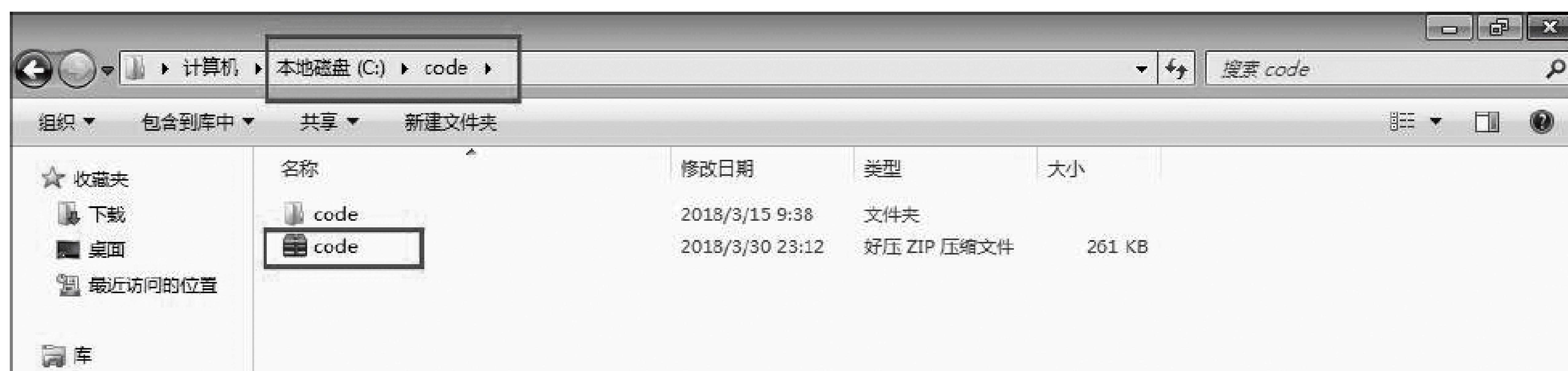


图 4-83 查看工程压缩文件

(3) 打开 Chrome 浏览器,输入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中,单击“高级”按钮。

(4) 在显示的隐藏折叠信息中,单击“继续前往 172.16.1.100”按钮。

(5) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(6) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(7) 在配置信息界面中,“任务名称”输入“溯源检测”,“开发语言”选中 Java 单选钮,“JDK 版本”选中 JDK1.7 单选钮,单击“上传文件”右侧的“浏览”按钮,选择“C:\code”目录下的 code.zip 文件,取消勾选“缺陷检测”复选框,取消勾选“合规检测”复选框,其他保持默认配置,单击“发起检测”按钮。

【实验预期】

可以通过代码安全保障系统发起溯源检测任务。

【实验结果】

(1) 发起检测后,系统返回任务列表界面。在这里可以查看“检测状态”“检测组件”和“危险组件数”等信息。当检测状态为“检测完成”时,单击最右侧的“缺陷审计”按钮。

(2) 可以看到危险组件为“commons-fileupload-1.3.jar”和检测结果等信息,如图 4-84 所示。

【实验思考】

(1) 尝试用代码安全保障系统发起合规检测,查看结果。

(2) 尝试用代码安全保障系统发起缺陷检测,查看结果。



图 4-84　查看结果(4.3.6)

4.3.7　Python 语言缺陷检测任务实验

【实验目的】

通过使用代码安全保障系统检测 Git 服务器中所存放的代码是否含有编码缺陷，了解代码安全保障系统的基本使用方法。

【知识点】

发起 Python 语言缺陷检测任务。

【场景描述】

A 公司研发部工程师小王编写的代码使用 Git 服务管理，需要使用代码安全保障系统对编写的 Python 代码进行安全测试，了解代码安全保障系统的基本使用方法。请帮助小王完成 Python 代码模块的快速检测。

【实验原理】

Git 是一个分布式版本控制系统，由 Linux 开源社区开发。此版本控制系统可以使项目的设计者将设计恢复到之前任一状态的选择权，这种选择权在设计过程无法进行下去时特别重要。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。
- Git 服务器：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 4-85 所示。

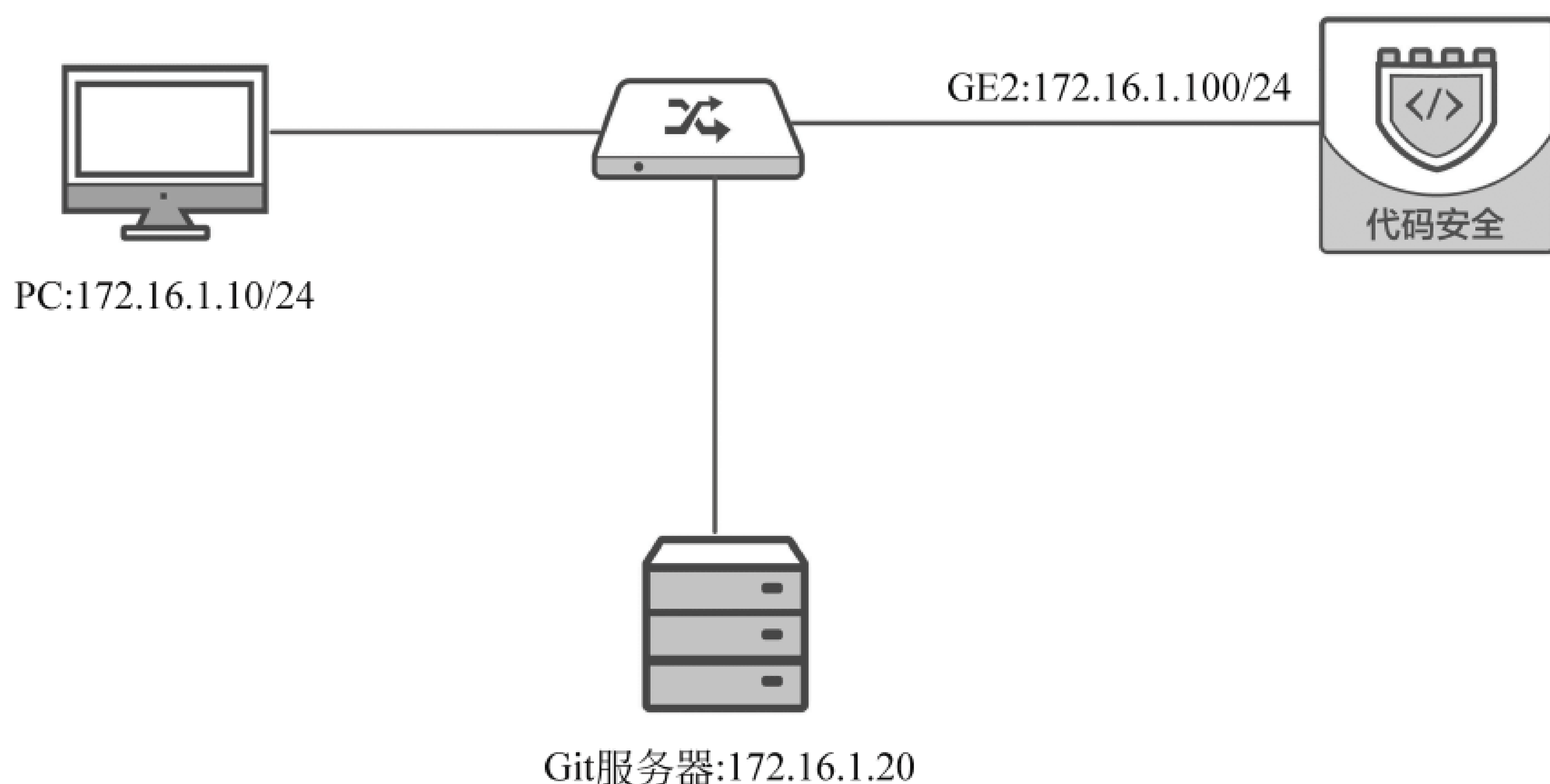


图 4-85　Python 语言缺陷检测任务实验拓扑图

【实验思路】

发起 Python 代码缺陷检测。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧 PC 虚拟机，如需登录密码，输入 123456。

(2) 进入"C:\code"目录，该目录下存有实验需要用到的 code.py 代码文件，如图 4-86 所示。

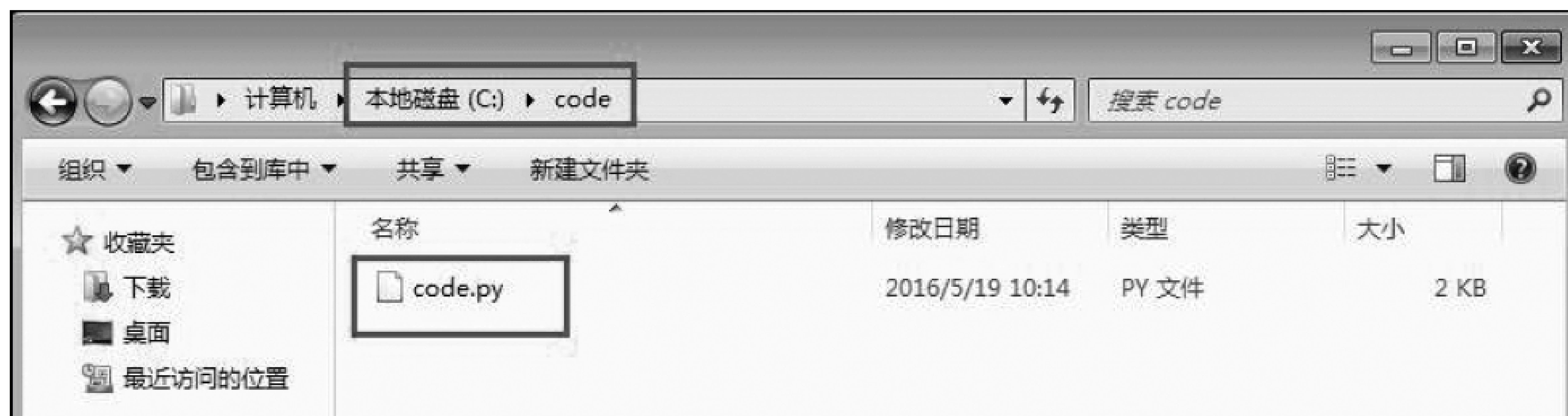


图 4-86　"code.py"文件

(3) 在"C:\code"目录下右击，在弹出的快捷菜单中选择"Git Bash Here"命令，进入 Git 命令行界面。

(4) 输入命令"git add code.py"，将 code.py 文件的信息添加到索引库中。

(5) 输入命令"git commit -m 'commit code.py'"，将索引库中的 code.py 文件提交到本地仓库中。

(6) 输入命令"git remote add origin http://172.16.1.20:1000/r/test.git"，添加远程仓库。

(7) 输入命令"git push -u origin master"，将本地仓库的内容提交到远程仓库 master 分支下。输入完毕后，输入远程仓库用户名 admin 密码 admin，单击 OK 按钮。

(8) 打开 Chrome 浏览器，输入代码卫士的 IP 地址"https://172.16.1.100"，在显示

的“您的连接不是私密连接”界面中单击“高级”按钮。

(9) 单击“继续前往 172.16.1.100(不安全)”按钮。

(10) 进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin,密码为“admin123!@#”),单击“登录”按钮。

(11) 进入代码卫士后,选择“快速检测”→“+发起快速检测”命令。

(12) 在配置信息界面中,“任务名称”输入“Python 语言缺陷检测”,“开发语言”选中 Python 单选钮,“源码来源”选中 Git 单选钮,“Git 地址”输入 http://172.16.1.20:1000/r/test.git,“Git 分支名称”默认为 master,“Git 用户名”输入 admin,“Git 密码”输入 admin,其他选择默认配置,单击“发起检测”按钮。

【实验预期】

代码安全保障系统可以检测出 Git 服务器中的代码缺陷。

【实验结果】

发起检测后,代码安全保障系统会从 Git 服务器中取出后缀为.py 类型的文件进行检测,检测完成后可以查看缺陷的数量以及等级(红色代表高危),如图 4-87 所示。



图 4-87　检测结果(4.3.7)

【实验思考】

(1) 如何根据代码卫士的修复建议修复缺陷?

(2) 如何导出检测结果?

第5章 代码安全保障系统检测结果审计

在前几章的基础上，我们掌握了如何在代码安全保障系统中进行缺陷检测、合规检测以及如何发起检测任务等。本章主要介绍如何对检测结果进行审计，包括对源代码的不同版本的检测结果进行比对分析，分析代码安全趋势；对比多个任务的缺陷密度、缺陷等级分布、缺陷类型分布等综合信息，以图形化报表的方式展现对比结果；按缺陷等级、缺陷类型进行分类统计；检测结果提供 Word、Excel 格式的检测报告，并可提供个性化的报告定制，导出检测结果。通过对检测结果进行以上全方位的审计，帮助管理者把握代码安全的整体情况。

5.1 检测结果分析

5.1.1 代码安全保障系统检测结果审计实验

【实验目的】

通过使用代码安全保障系统对编写的代码进行缺陷检测，根据修复建议对缺陷等级和状态进行修改，使管理员掌握使用代码安全保障系统进行代码审计的方法。

【知识点】

检测结果审计。

【场景描述】

A 公司研发部工程师小王编写了一个代码模块，使用代码安全保障系统对编写的代码进行缺陷检测，检测完成后，对检测出的结果进行查看、修改缺陷等级和状态。请帮助小王完成检测结果的审计。

【实验原理】

导入危险文件是指程序中利用 include()、include_once()、require()和 require_once()这 4 个文件包含函数来引用其他文件中的程序。当引入的文件用户可控时，若未对文件名称进行严格过滤或者过滤机制被绕过，容易引起文件泄露和恶意代码注入，此漏洞被称为导入危险文件。

代码安全保障系统能够检测出代码中的缺陷，并给出详细描述以及修复建议，方便开

发人员检测出源代码中的缺陷并进行有针对性的修复。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 5-1 所示。

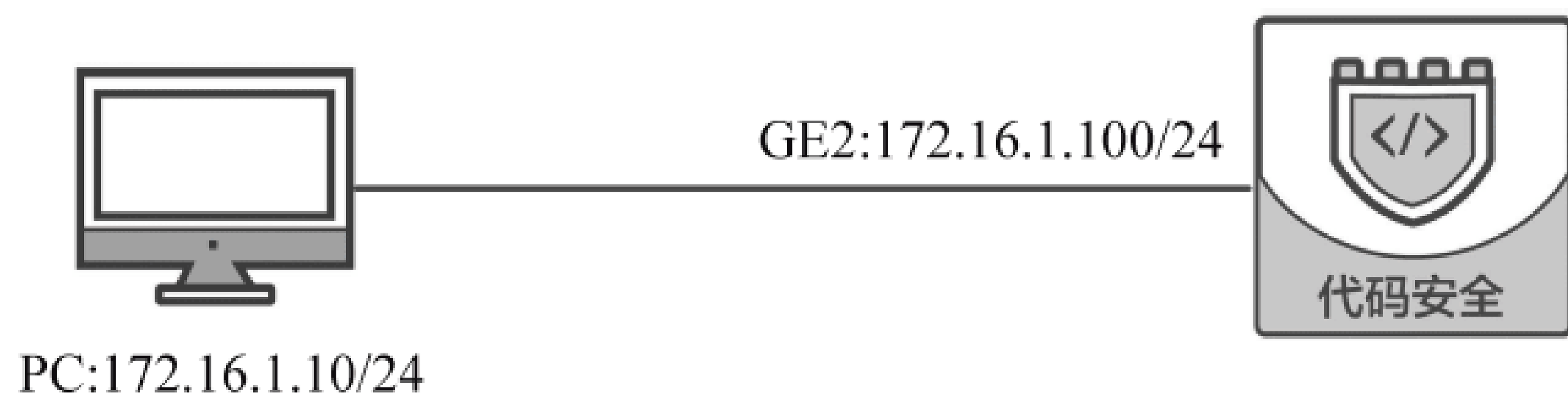


图 5-1 代码安全保障系统检测结果审计实验拓扑图

【实验思路】

（1）发起 PHP 代码缺陷检测。

（2）根据修改建议对代码进行修改。

（3）对修改后的 PHP 代码发起缺陷检测，检测修改效果。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）打开“C:\code”目录下的 code.php 文件。

（3）该代码通过 include()函数来包含用户指定的文件，从代码中可以看出程序未对文件名进行过滤，容易造成信息泄露或恶意代码注入，如图 5-2 所示。



图 5-2 代码解析(5.1.1)

（4）打开 Chrome 浏览器，输入代码卫士的网址“https://172.16.1.100”，在显示的“您的连接不是私密连接”界面中单击“高级”按钮。

（5）单击“继续前往 172.16.1.100(不安全)”按钮。

（6）进入代码卫士登录界面。输入用户名和密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

（7）进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

（8）在配置信息界面中，“任务名称”输入“检测结果审计”，“开发语言”选中 PHP 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，其他保

持默认配置，单击“发起检测”按钮。

【实验预期】

(1) 界面显示代码卫士检测出的代码缺陷。

(2) 修改后的代码再次被检测后缺陷消除。

【实验结果】

1. 界面显示代码卫士检测出的代码缺陷

(1) 检测完成后可以看到代码卫士从此代码模块中检测出一个缺陷，如图 5-3 所示。



图 5-3　检测结果(5.1.1)

(2) 单击“缺陷审计”按钮(图 5-3 最右框位置)。可以看到代码卫士将该缺陷的风险等级定义为高。单击左侧框中的“＋”按钮，展开目录。

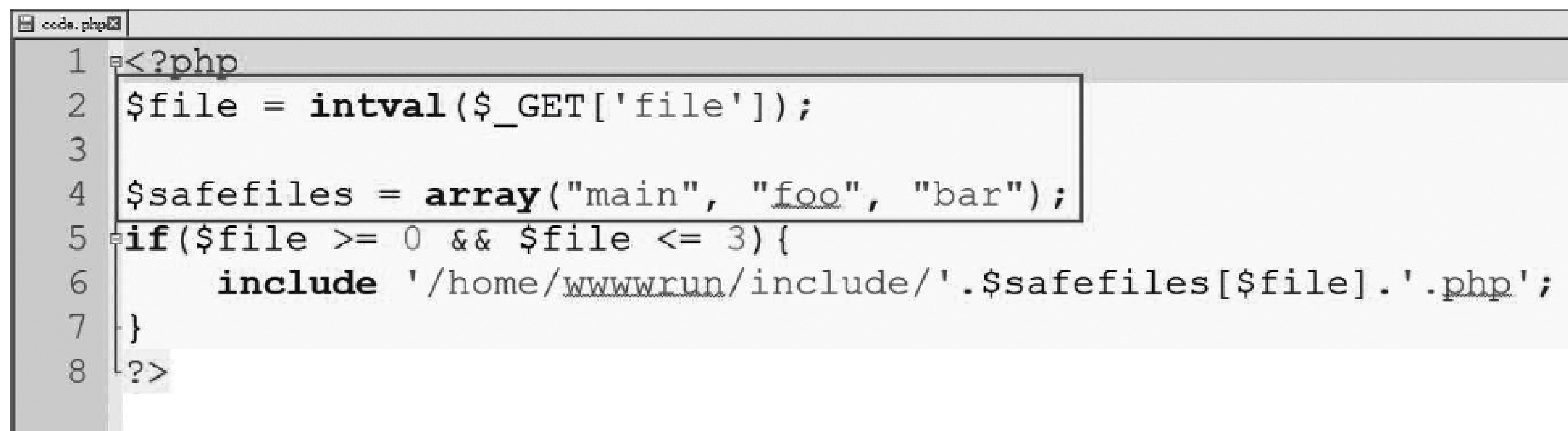
(3) 选择左侧的“code.php(3)”命令，可以快速定位有问题的代码。

(4) 选择“详细信息”命令，可以查看问题代码中该条缺陷的具体信息。

(5) 选择“修复建议”命令，查看代码卫士给的修复建议。

2. 修改后的代码再次被检测后缺陷消除

(1) 最小化浏览器窗口，打开“C:\code”目录下的 code.php 文件，根据修复建议对代码进行修改，采用文件列表的方法来限制用户包含的文件，相关代码如图 5-4 所示。



```
1 <?php
2 $file = intval($_GET['file']);
3
4 $safefiles = array("main", "foo", "bar");
5 if($file >= 0 && $file <= 3){
6     include '/home/wwwrun/include/'.$safefiles[$file].'.php';
7 }
8 ?>
```

图 5-4　代码修改(5.1.1)

(2) 右击 code.php，在弹出的快捷菜单中选择“添加到压缩文件(A)…”命令。

(3) “压缩到”中的文件名称输入“code-fix.zip”，单击“立即压缩”按钮。

(4) 打开 Chrome 浏览器，在之前的界面上选择“快速检测”→“＋发起快速检测”命令。

(5) 在配置信息界面中,“任务名称”输入“检测结果审计修复检测”,“开发语言”选中PHP单选钮,“源码来源”设置为“本地”,“上传文件”设置为“C:\code”目录下的“code-fix.zip”文件,其他保持默认配置,单击“发起检测”按钮。

(6) 在任务列表界面等待检测完成,可以看到该项目检测完成后,“等级分布”的图形示意为空,缺陷总数为0,如图5-5所示。

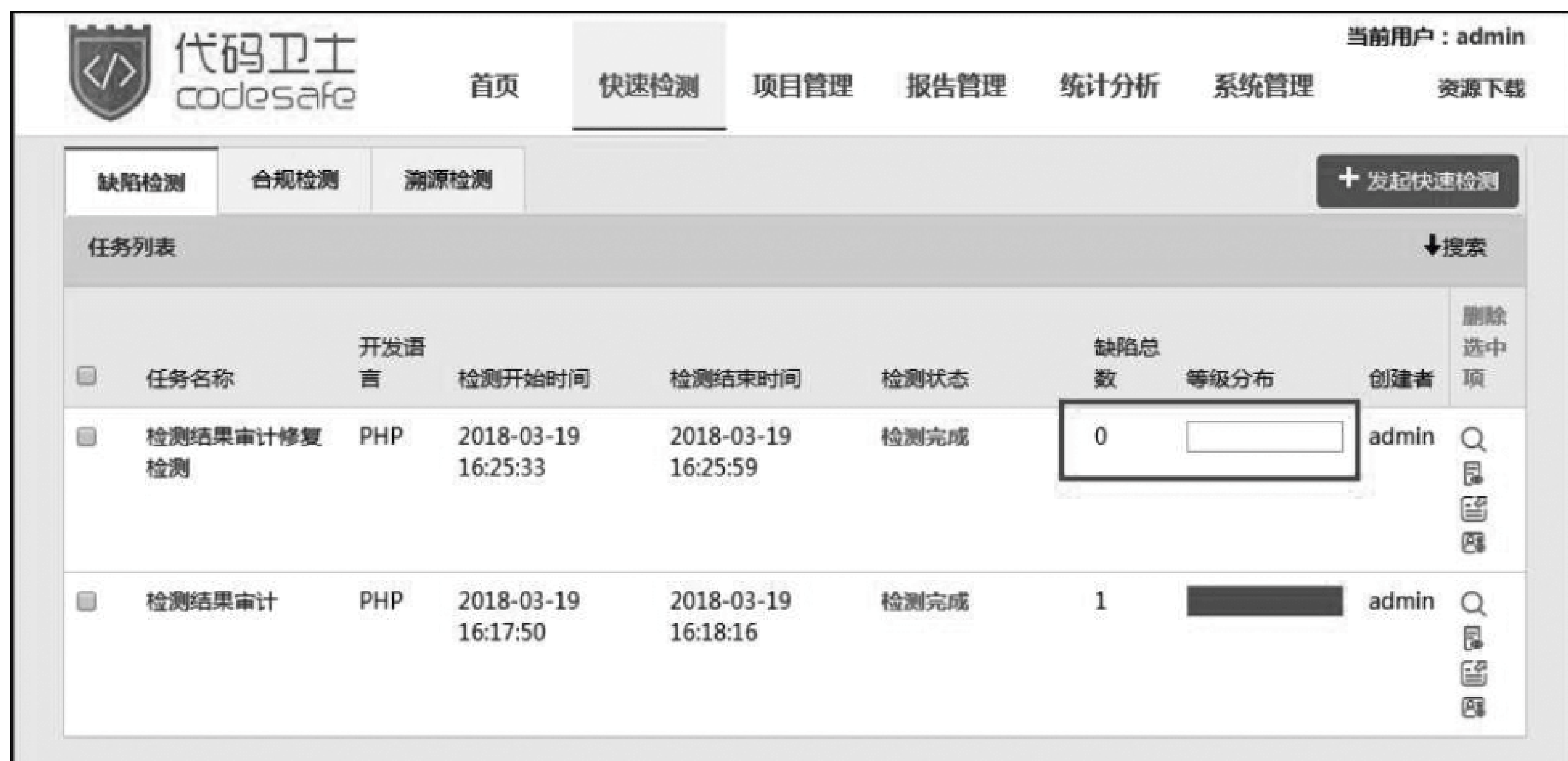


图5-5 检测完成(5.1.1)

【实验思考】

(1) 搭建Web服务器网站,使用实验中的源码尝试包含一个本地文件。

(2) 搭建Web服务器网站,使用实验中的源码尝试包含一个远端服务器文件。

5.1.2 代码安全保障系统检测结果mybug统计实验

【实验目的】

管理员可以熟练掌握代码卫士系统的用户管理部分,包括新建部门、用户以及分配权限等。普通用户熟练掌握按条件搜索功能,比如按任务名称搜索、按创建者所搜等。

【知识点】

Java代码缺陷检测。

【场景描述】

A公司研发部有多名工程师,使用代码安全保障系统对研发部的代码进行缺陷检测,检测完成后,每名工程师只希望看到自己提交的bug引发的安全缺陷。

请完成mybug的查看。

【实验原理】

用户可将代码上传到代码卫士安全管理中心进行检测,既可以通过浏览器客户端直接查看结果,也可以通过搜索功能来筛选检测结果。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 5-6 所示。

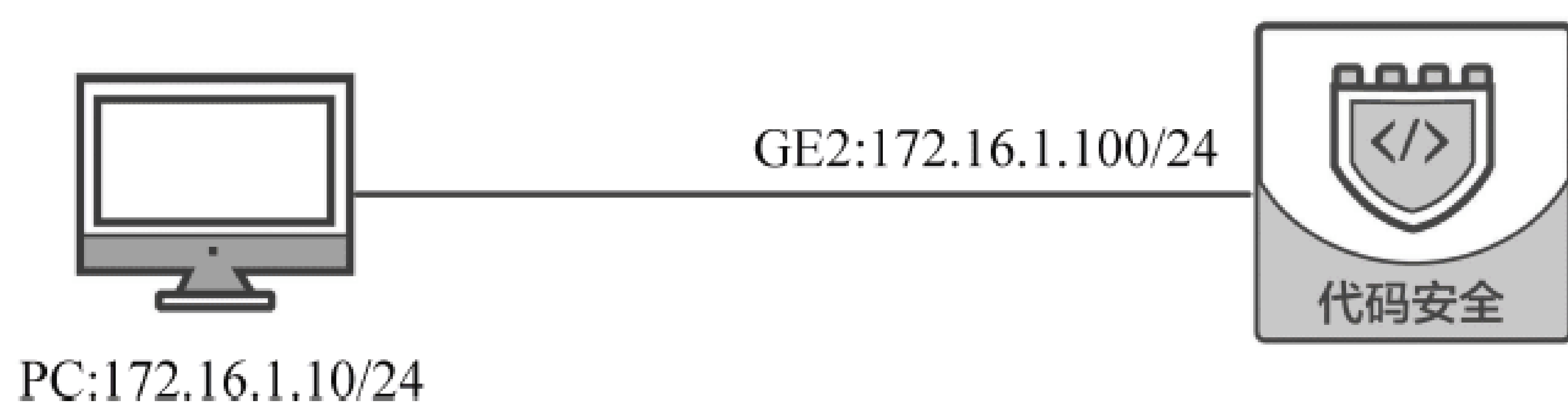


图 5-6　代码安全保障系统检测结果 mybug 统计实验拓扑图

【实验思路】

(1) 添加部门、用户和角色。

(2) 多用户发起快速检测。

(3) 搜索查看结果。

【实验步骤】

(1) 进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

(2) 查看“C:\code”目录下的 code.zip 文件。

(3) 打开 Chrome 浏览器，进入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(4) 在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

(5) 跳转到登录界面，输入用户名与密码(默认用户名为 admin，密码为“admin123!@#”)，单击“登录”按钮。

(6) 登录成功后，单击“系统管理”按钮。

(7) 选择“用户管理”→“部门”→“+添加部门”命令。

(8) 在添加部门界面中，“部门名称”输入“研发部”，“描述”输入“上传代码并检测”，单击“保存”按钮。

(9) 查看新添加的研发部，单击“角色”按钮。

(10) 代码卫士系统默认含有两个角色，用户和管理员，单击“+添加角色”按钮。

(11) 在添加角色界面中，“角色名称”输入“上传代码并检测”，“权限”设置为“快速检测”。

(12) 保存后即可查看新添加的角色，单击“用户”按钮。

(13) 单击“+添加用户”按钮，添加新的用户。

(14) 在添加用户界面中，“用户名”输入 test1，“密码”和“确认密码”输入“test123!@#”，“部门”设置为“研发部”，“角色”设置为“上传代码并检测”，单击“保存”按钮。

(15) 保存后系统返回用户列表界面，单击“＋添加用户”按钮。

(16) 在添加用户界面中，“用户名”输入 test2，“密码”和“确认密码”输入“test123!@#”，“部门”设置为“研发部”，“角色”设置为“上传代码并检测”，单击“保存”按钮。

(17) 保存后系统返回用户列表界面，单击右上角的 admin 按钮，在显示的列表中单击“退出”按钮。

(18) 跳转到登录界面，输入用户名与密码(输入用户名 test1，密码为“test123!@#”)，单击“登录”按钮。

(19) 进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

(20) 在配置信息界面中，“任务名称”输入“检测结果 mybug 统计”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框。单击“指定人员”按钮。

(21) 勾选“研发部”复选框，单击“保存”按钮，如图 5-7 所示。

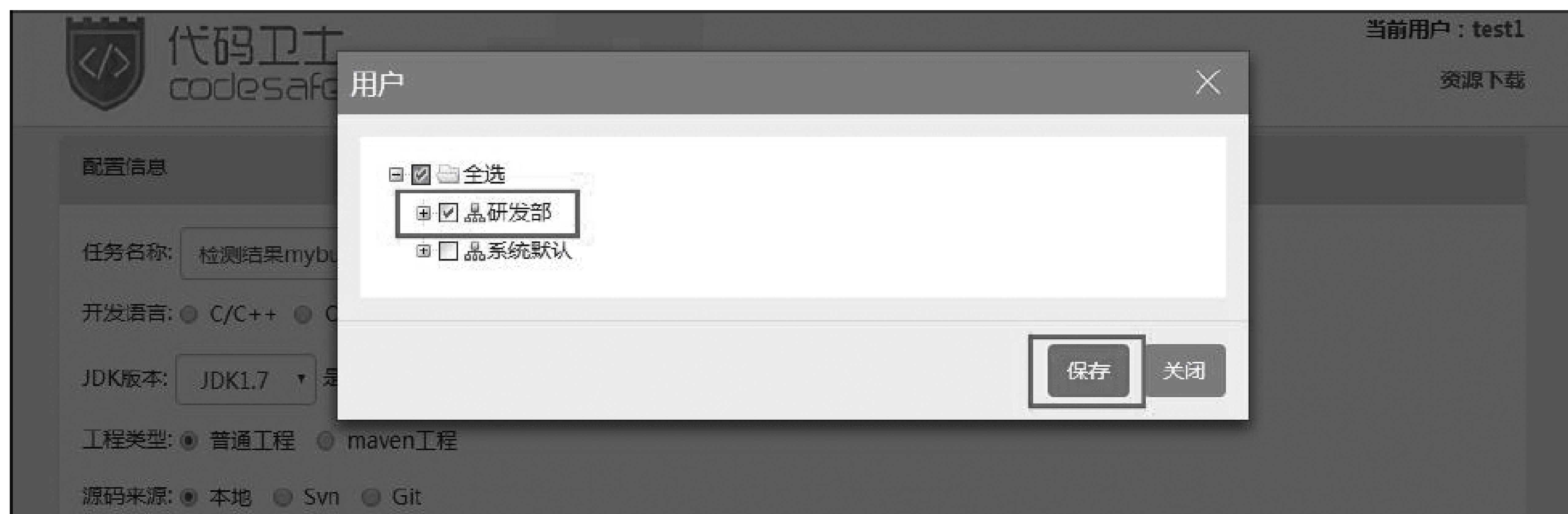


图 5-7　选择指定人员

(22) 单击“发起检测”按钮。

(23) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”“等级分布”和“创建者”等信息。当检测状态为“检测完成”时，单击右上角的 test1 按钮。

(24) 单击“退出”按钮，退出系统。

(25) 跳转到登录界面，输入用户名与密码(用户名为 test2，密码为“test123!@#”)，单击“登录”按钮。

(26) 进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

(27) 在配置信息界面中，“任务名称”输入“检测结果 mybug 统计”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框。单击“指定人员”按钮。

(28) 勾选“研发部”复选框，单击“保存”按钮。

(29) 单击“发起检测”按钮。

(30) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”

和“等级分布”等信息。

【实验预期】

可以只查看自己提交的 bug 引发的安全缺陷。

【实验结果】

(1) 单击“搜索”按钮，如图 5-8 所示。



图 5-8　单击“搜索”按钮

(2) 在配置信息界面中，“创建者”输入 test2。单击“搜索”按钮，如图 5-9 所示。



图 5-9　配置信息

(3) 可以看到只有自己提交的 bug 引发的安全缺陷，如图 5-10 所示。

【实验思考】

(1) 尝试按任务名称搜索，查看结果。



图 5-10　搜索结果(5.1.2)

(2) 尝试按检测时间搜索,查看结果。

5.1.3　代码安全保障系统检测结果统计分析实验

【实验目的】

通过使用代码安全保障系统检测上传的代码模块是否存在缺陷,并对检测出的缺陷进行统计分析。

【知识点】

统计分析。

【场景描述】

A 公司研发部工程师小王编写了一个代码模块,使用代码安全保障系统对编写的代码进行缺陷检测,检测完成后,对检测出的缺陷进行统计分析。

请帮助小王完成检测结果的统计分析。

【实验原理】

用户可以按照项目、部门、用户、类型等维度对项目情况进行多角度统计分析。

【实验设备】

- 安全设备:代码安全保障系统 1 套。
- 主机终端:Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 5-11 所示。

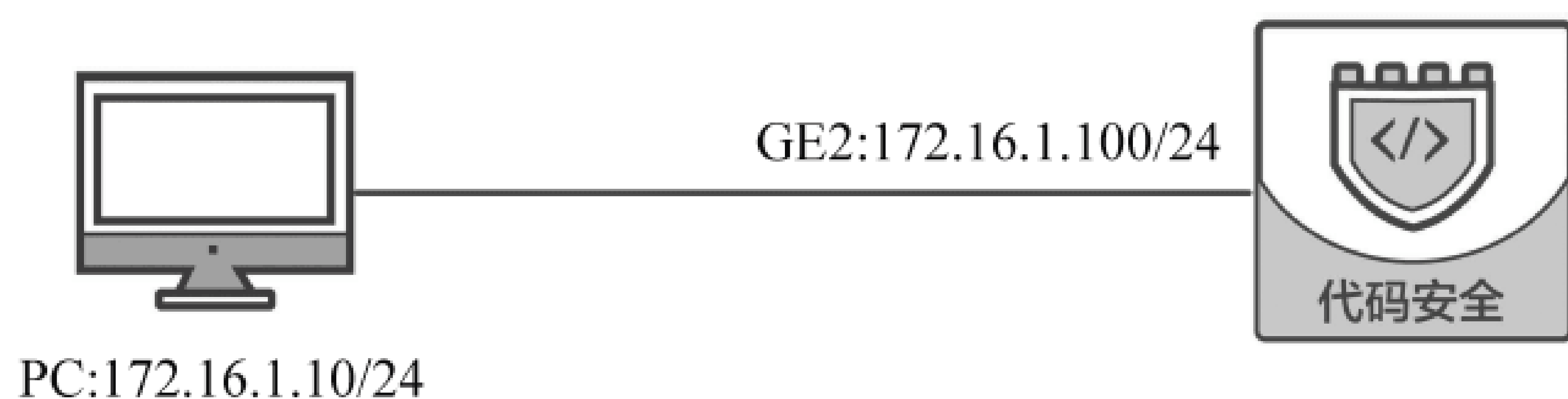


图 5-11　代码安全保障系统检测结果统计分析实验拓扑图

【实验思路】

（1）发起 Java 代码缺陷检测。

（2）对检测出的缺陷进行统计分析。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）查看“C:\code”目录下的 code.zip 文件。

（3）打开 Chrome 浏览器，输入代码卫士登录界面网址“https://172.16.1.100”。在显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

（4）在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

（5）跳转到登录界面，输入用户名与密码（默认用户名为 admin，密码为“admin123!@#”），单击“登录”按钮。

（6）进入代码卫士后，选择“快速检测”→“+发起快速检测”命令。

（7）在配置信息界面中，“任务名称”输入“检测结果统计分析”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的“code.zip”文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

（8）发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息，如图 5-12 所示。



图 5-12　开始检测

【实验预期】

可以按照缺陷类别对检测出的缺陷生成统计分析图。

【实验结果】

(1) 选择“统计分析”命令,如图 5-13 所示。



图 5-13 统计分析

(2) 选择“按类型统计”命令。

(3) 在配置信息界面中,“开发语言”选中 Java,“缺陷分类”设置为“全选”,取消勾选“项目管理”复选框,其他保持默认配置,单击“开始统计”按钮,如图 5-14 所示。



图 5-14 配置信息

(4) 统计结果如图 5-15 所示。

【实验思考】

(1) 尝试按照代码注入质量进行统计结果分析。

(2) 尝试按照代码注入进行统计结果分析。

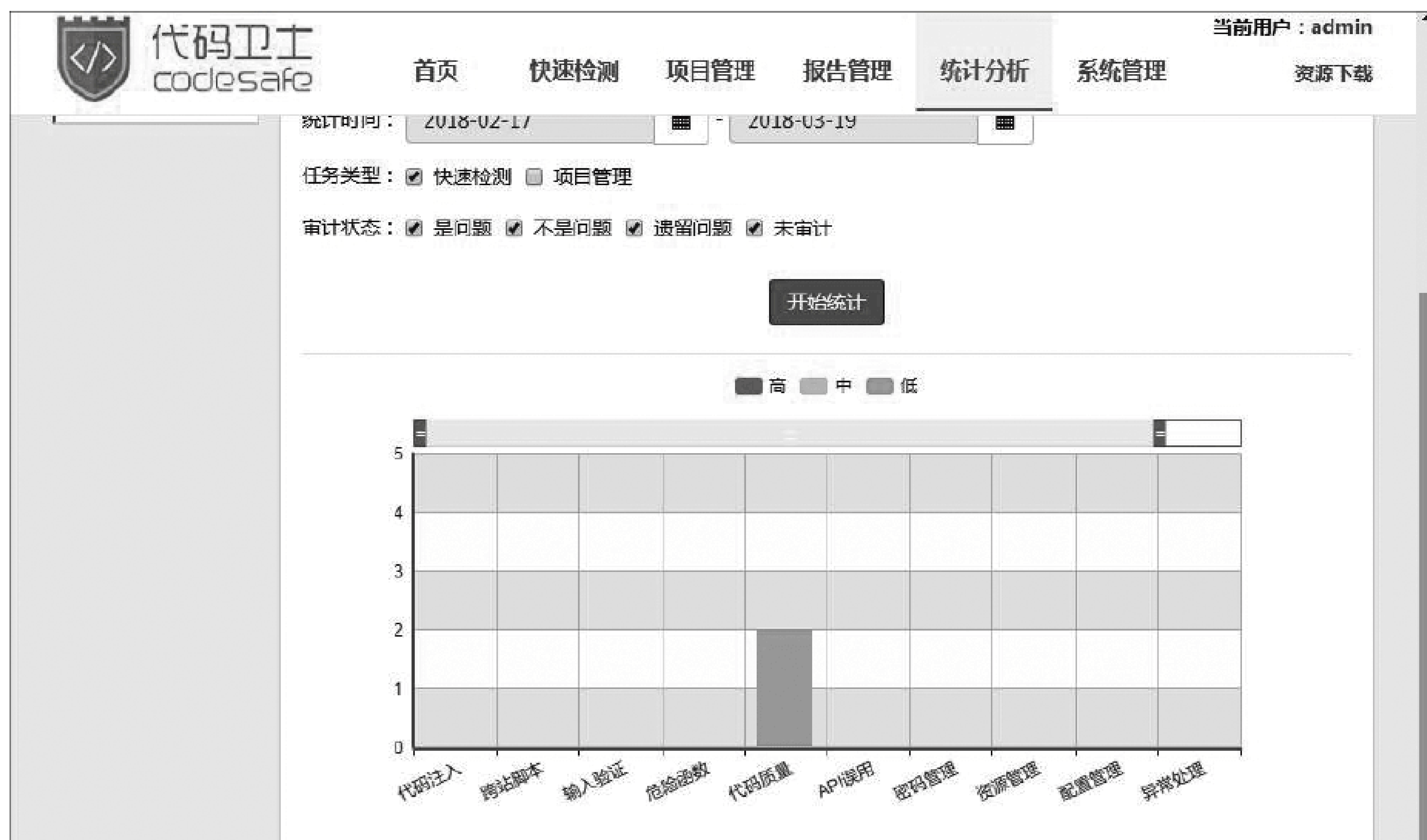


图 5-15　统计结果

5.2　检测结果管理

5.2.1　代码安全保障系统审计信息携带实验

【实验目的】

通过代码安全保障系统对编写的代码进行缺陷检测，检测完成后，对检测出的缺陷进行审计，再发起该代码的检测时，携带上次检测结果的审计信息。

【知识点】

审计信息携带。

【场景描述】

A 公司研发部工程师小王编写的代码，使用代码安全保障系统对编写的代码进行缺陷检测，检测完成后，对检测出的缺陷进行审计，再发起该代码的检测时，携带上次检测结果的审计信息。

请帮助小王完成检测结果的审计信息携带。

【实验原理】

使用者可以直接选择代码卫士安全管理中心的代码进行源代码缺陷或合规检测，并进行人工审计，将检测结果导入缺陷管理系统供开发人员修改。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 5-16 所示。

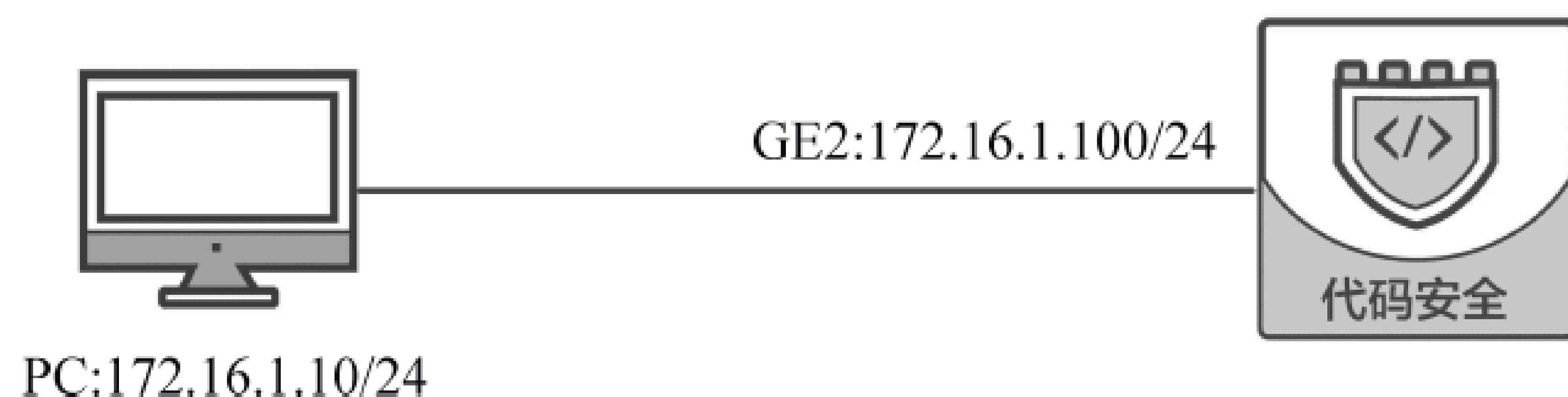


图 5-16 代码安全保障系统审计信息携带实验拓扑图

【实验思路】

(1) 发起 Java 代码缺陷检测。

(2) 对检测出的缺陷进行审计。

(3) 再次发起 Java 代码缺陷检测,携带上次检测结果的审计信息。

【实验步骤】

(1) 进入实验平台对应的实验拓扑,登录左侧的 PC 虚拟机,如需登录密码,输入 123456。

(2) 查看"C:\code"目录下的 code.zip 文件。

(3) 打开 Chrome 浏览器,输入代码卫士登录界面网址"https://172.16.1.100"。在显示的"您的连接不是私密连接"界面中,单击"高级"按钮。

(4) 在显示的隐藏折叠信息中,单击"继续前往 172.16.1.100"按钮。

(5) 跳转到登录界面,输入用户名与密码(默认用户名为 admin,密码为"admin123!@#"),单击"登录"按钮。

(6) 进入代码卫士,选择"快速检测"→"+发起快速检测"命令。

(7) 在配置信息界面中,"任务名称"输入"审计信息携带","开发语言"选中 Java 单选钮,"JDK 版本"选中 JDK1.7 单选钮,单击"上传文件"右侧的"浏览"按钮,选择"C:\code"目录下的 code.zip 文件,取消勾选"合规检测"复选框,取消勾选"溯源检测"复选框,其他保持默认配置,单击"发起检测"按钮。

(8) 发起检测后,系统返回任务列表界面。在这里可以查看"检测状态""缺陷总数"和"等级分布"等信息。

(9) 可以看到代码中存在资源未释放的位置有两处,分别为"code.java(72)"和"code.java(73)",下面以"code.java(72)"来说明。选择"code.java(73)"命令,可以快速定位有问题的代码。

(10) 单击"缺陷审计"按钮。

(11) 在缺陷审计配置信息界面中,"状态"选中"是问题"单选钮,其他保持默认配置,单击"提交"按钮,如图 5-17 所示。

【实验预期】

再次发起检测后,可以查看到上次审计检查结果的审计信息。



图 5-17　提交审计

【实验结果】

（1）选择“快速检测”→“＋发起快速检测”命令。

（2）在配置信息界面中，“任务名称”输入“审计信息携带标记”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，“是 J2EE 工程”选中“否”单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，“是否携带”设置为“是”，“任务列表”设置为“审计信息携带”，其他保持默认配置，单击“发起检测”按钮。

（3）在任务列表界面等待检测完成，单击最右侧的“缺陷审计”按钮。

（4）可以看到代码卫士检测列出了相关缺陷，并判断该缺陷的风险等级为“中”。单击左侧框中的“＋”按钮，展开目录，如图 5-18 所示。



图 5-18　展开界面

(5) 选择"code.java(72)"命令。

(6) 单击"缺陷审计"按钮,如图 5-19 所示。



图 5-19　单击"缺陷审计"按钮

(7) 可以看到上次审计检查结果的审计信息,如图 5-20 所示。



图 5-20　查看审计信息

【实验思考】

(1) 尝试将缺陷审计中的状态改为高,查看结果。

（2）尝试填写缺陷审计中的备注，查看结果。

5.2.2　代码安全保障系统导出检测结果实验

【实验目的】

通过使用代码安全保障系统检测上传的代码模块是否存在的缺陷，检测完成后，并将检测出的缺陷导出到 Word 文档查看。

【知识点】

报告导出。

【场景描述】

A 公司研发部工程师小王编写了一个代码模块，使用代码安全保障系统对编写的代码进行缺陷检测，检测完成后，对检测出的缺陷导出到 Word 文档查看，请帮助小王完成检测结果的导出。

【实验原理】

使用者通过浏览器可直接将待检软件源代码上传至安全管理中心，进行源代码缺陷检测。检测结果可以导出为 Word 或 Excel 格式的报表。

【实验设备】

- 安全设备：代码安全保障系统 1 套。
- 主机终端：Windows 7 SP1 主机 1 台。

【实验拓扑】

实验拓扑如图 5-21 所示。

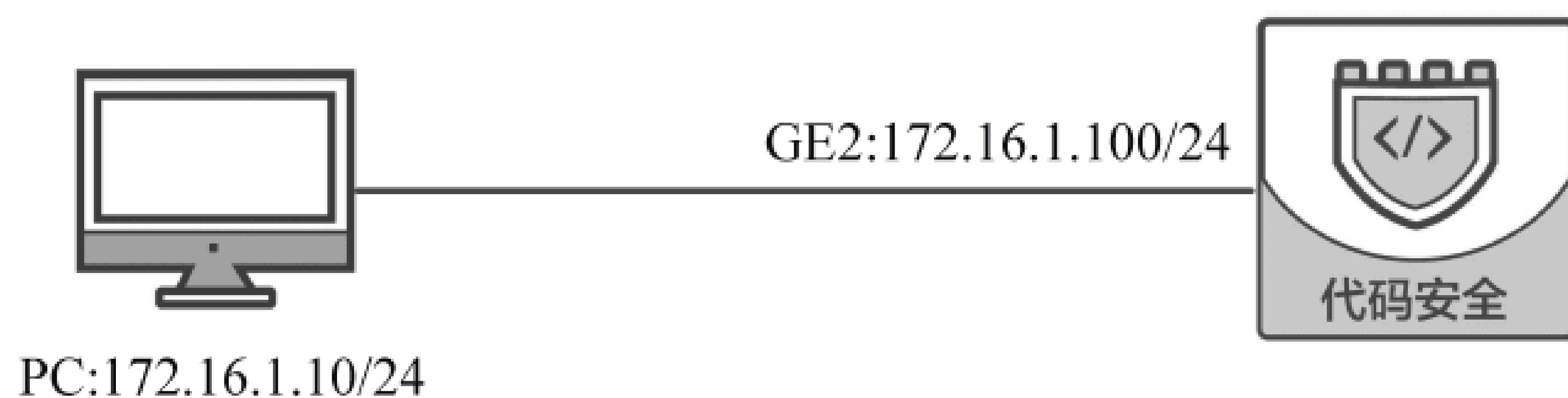


图 5-21　代码安全保障系统导出检测结果实验拓扑图

【实验思路】

（1）发起 Java 代码缺陷检测。

（2）对检测出的缺陷导出 Word 文档查看。

【实验步骤】

（1）进入实验平台对应的实验拓扑，登录左侧的 PC 虚拟机，如需登录密码，输入 123456。

（2）查看“C:\code”目录下的 code.zip 文件。

（3）打开 Chrome 浏览器，输入代码卫士登录界面网址“https://172.16.1.100”。在

显示的“您的连接不是私密连接”界面中，单击“高级”按钮。

(4) 在显示的隐藏折叠信息中，单击“继续前往 172.16.1.100”按钮。

(5) 跳转到登录界面，输入用户名与密码（默认用户名为 admin，密码为“admin123!@#”），单击“登录”按钮。

(6) 进入代码卫士后，选择“快速检测”→“＋发起快速检测”命令。

(7) 在配置信息界面中，“任务名称”输入“导出检测结果”，“开发语言”选中 Java 单选钮，“JDK 版本”选中 JDK1.7 单选钮，单击“上传文件”右侧的“浏览”按钮，选择“C:\code”目录下的 code.zip 文件，取消勾选“合规检测”复选框，取消勾选“溯源检测”复选框，其他保持默认配置，单击“发起检测”按钮。

(8) 发起检测后，系统返回任务列表界面。在这里可以查看“检测状态”“缺陷总数”和“等级分布”等信息，如图 5-22 所示。



图 5-22　开始检测

【实验预期】

可以将检测出的缺陷导出到 Word 文档。

【实验结果】

单击“导出报告”按钮，可以看到检测报告导出成功，如图 5-23 所示。



图 5-23　导出报告成功

【实验思考】

(1) 尝试导出 PDF 格式的报告。

(2) 尝试导出 Excel 格式的报告。